3/5/1 (Item 1 from file: 351)

DIALOG(R)File 351:Derwent WPI

010898234 **Image available**
WPI Acc No: 1996-395185/199640
XRPX Acc No: N96-333032

Debugger system for debugging distributed target application system - has debugger GUI and one or more debugger engines which may reside on local or remote host computer

Patent Assignee: SUN MICROSYSTEMS INC (SUNM)
Inventor: DAVIDSON A E; MASAMITSU J A
Number of Countries: 008 Number of Patents: 005
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
EP 730227	A1	19960904	EP 96301430	A	19960301	199640	B
CA 2170724	A	19960904	CA 2170724	A	19960229	199701	
JP 9120366	A	19970506	JP 9673240	A	19960304	199728	
US 5819093	A	19981006	US 95399120	A	19950303	199847	
US 6042614	A	20000328	US 95399120	A	19950303	200023	
			US 985287	A	19980109		

Priority Applications (No Type Date): US 95399120 A 19950303; US 985287 A 19980109
Cited Patents: US 4589068; US 5371746
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
EP 730227	A1	E	28	G06F-011/00	
Designated States (Regional): DE FR GB IT SE					
US 6042614	A			G06F-009/445	Cont of application US 95399120
JP 9120366	A		24	G06F-011/28	
CA 2170724	A			G06F-009/44	
US 5819093	A			G06F-009/455	

Abstract (Basic): EP 730227 A

The distributed debugger system includes a debugger GUI (502) which provides an interface with one or more debugger engines (dbx engines) for communicating with the dbx engines which reside on local and remote host computers. A communications mechanism is used by the dbx engines and the debugger GUI for sending and receiving messages from each other.

A remote dbx engine resides on a host computer and is connected to the debugger GUI by the communications mechanism. request and replies are made through an object request broker (ORB) that is aware of the locations and status of objects in the distributed object environment.

USE - Debugging new applications using objects in widely distributed, object-oriented, client server system.

ADVANTAGE - Enables developer to use objects and object implementations located on different host machine which is unknown to developer.

Dwg.14/15

Title Terms: SYSTEM; DEBUG; DISTRIBUTE; TARGET; APPLY; SYSTEM; ONE; MORE; ENGINE; LOCAL; REMOTE; HOST; COMPUTER
Derwent Class: T01
International Patent Class (Main): G06F-009/44; G06F-009/445; G06F-009/455; G06F-011/00; G06F-011/28
International Patent Class (Additional): G06F-009/06; G06F-012/00; G06F-015/16
File Segment: EPI

3/5/2 (Item 1 from file: 347)

DIALOG(R)File 347:JAPIO

05505566
SYSTEM AND METHOD FOR DISTRIBUTED DEBUGGING OF DEBUGGING OF DISTRIBUTED APPLICATION PROGRAM

PUB. NO.: 09-120366 **JP 9120366** A]
PUBLISHED: May 06, 1997 (19970506)

INVENTOR(s): ANDORIYUU II DEBITSUDOSON
JIYON EI MASAMITSU
APPLICANT(s): SUN MICROSYST INC [198211] (A Non-Japanese Company or Corporation), US (United States of America)

APPL. NO.: 08-073240 [JP 9673240]
FILED: March 04, 1996 (19960304)
PRIORITY: 7-399,120 [US 399120-1995], US (United States of America), March 03, 1995 (19950303)
INTL CLASS: [6] G06F-011/28; G06F-009/06; G06F-009/44; G06F-012/00; G06F-015/16
JAPIO CLASS: 45.1 (INFORMATION PROCESSING -- Arithmetic Sequence Units); 45.2 (INFORMATION PROCESSING -- Memory Units); 45.4 (INFORMATION PROCESSING -- Computer Applications)

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報**（Ａ）　(11)特許出願公開番号

特開平9－120366

(43)公開日　平成９年(1997)５月６日

(51)Int.Cl.[6]		識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ	11/28		7313－5Ｂ	Ｇ０６Ｆ	11/28	Ａ
	9/06	５３０			9/06	５３０Ｑ
	9/44	５３５			9/44	５３５
	12/00	５４５			12/00	５４５Ａ
	15/16	３７０			15/16	３７０Ｎ

審査請求　未請求　請求項の数25　ＦＤ　（全 24 頁）

(21)出願番号　特願平8－73240

(22)出願日　平成８年(1996)３月４日

(31)優先権主張番号　０８／３９９１２０

(32)優先日　1995年３月３日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　591064003
サン・マイクロシステムズ・インコーポレーテッド
SUN MICROSYSTEMS, INCORPORATED
アメリカ合衆国　94043　カリフォルニア州・マウンテンビュー・ガルシア　アヴェニュウ・2550

(72)発明者　アンドリュー・イー・デビッドソン
アメリカ合衆国　95006　カリフォルニア州・ブルダー　クリーク・ホプキンス　ガルチ・1222

(74)代理人　弁理士　山川　政樹

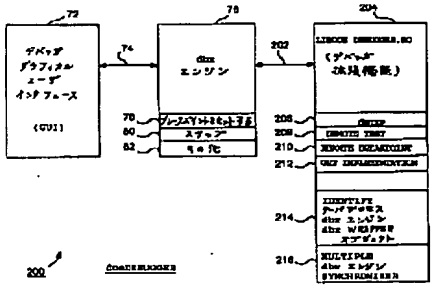
最終頁に続く

(54)【発明の名称】　分散アプリケーション・プログラムをデバッグする分散デバッグのためのシステムおよび方法

(57)【要約】　（修正有）

【課題】　分散アプリケーションのデバッグを容易に行えるようにする。

【解決手段】　アプリケーションのプログラマ／開発者が１台のホストマシンのところにおり、開発されるアプリケーションがプログラマ／開発者には未知の他のホストマシンにおかれていることのあるオブジェクトおよびオブジェクト・インプリメンテーションを利用する分散ターゲット・コンピュータ・アプリケーションをデバッグする分散デバッガ・システムにおいて、分散オブジェクト環境における要求及び応答は、オブジェクトの場所及び状況を認識しているオブジェクト・リクエスト・ブローカ（ＯＲＢ）によって行われ、ＯＲＢを実施するのに適している１つのアーキテクチャが、共通オブジェクト・リクエスト・ブローカ・アーキテクチャ仕様によって与えられる。



1

【特許請求の範囲】

【請求項１】　ローカル・ホスト・コンピュータおよび１つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムにおいて、

デバッガＧＵＩおよび１つまたは複数のデバッガ・エンジン（以下、「ｄｂｘエンジン」と呼ぶ）であって、前記デバッガＧＵＩが前記ｄｂｘエンジンと通信し、かつ前記デバッガ・システムのユーザと通信するためのインタフェース機構を備えており、前記ｄｂｘエンジンが前記ローカルおよび遠隔のコンピュータに常駐できるデバッガＧＵＩおよび１つまたは複数のｄｂｘエンジンと、メッセージを互いに送受信するために前記ｄｂｘエンジンと前記デバッガＧＵＩが使用する通信機構と、

前記の１つまたは複数のｄｂｘエンジンの１つであり、前記ローカル・ホスト・コンピュータから離隔したホスト・コンピュータにあり、前記通信機構によって前記デバッガＧＵＩに接続されている遠隔ｄｂｘエンジンとを備えている分散デバッガ・システム。

【請求項２】　ローカル・ホスト・コンピュータおよび１つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムにおいて、

デバッガＧＵＩおよび１つまたは複数のデバッガ・エンジン（以下、「ｄｂｘエンジン」と呼ぶ）であって、前記デバッガＧＵＩが前記ｄｂｘエンジンと通信し、かつ前記デバッガ・システムのユーザと通信するためのインタフェース機構を備えており、前記ｄｂｘエンジンが前記ローカルおよび遠隔のコンピュータに常駐できるデバッガＧＵＩおよび１つまたは複数のｄｂｘエンジンと、メッセージを互いに送受信するために前記ｄｂｘエンジンと前記デバッガＧＵＩが使用する通信機構と、

遠隔ホスト・コンピュータにある前記ターゲット・アプリケーション・システムの一部をデバッグする際に使用される新しいｄｂｘエンジンを前記遠隔ホスト・コンピュータに作成するために前記デバッガＧＵＩが使用するｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙ機構とを備えている分散デバッガ・システム。

【請求項３】　前記ｄｂｘエンジンの１つが前記ｄｂｘエンジンの他のものと、これらのｄｂｘエンジンが異なるホスト・コンピュータにあるかどうかにかかわりなく、通信することを可能とする第１の通信機構をさらに含んでいることを特徴とする、請求項２に記載の分散デバッガ・システム。

【請求項４】　前記デバッガＧＵＩが前記ｄｂｘエンジンの１つに、前記ｄｂｘエンジンがどのホスト・コンピュータにあるかにかかわりなく、フォーカスすることを可能とする第２の通信機構をさらに含んでいることを特徴とする、請求項２に記載の分散デバッガ・システム。

【請求項５】　前記ユーザが、前記ｄｂｘエンジンがどのホスト・コンピュータにあるかにかかわりなく、すべての活動ｄｂｘエンジンのリストを前記デバッガＧＵＩから取得できるようにする第３の通信機構をさらに含んでいることを特徴とする、請求項２に記載の分散デバッガ・システム。

【請求項６】　ローカル・ホスト・コンピュータおよび１つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用するｄｂｘエンジンにおいて、前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続するが、前記ターゲット・アプリケーション・システム自体の一部ではない中間インタフェース定義言語（「ＩＤＬ」）生成コード機構を無視する（すなわち、「ステップ・オーバーする」）ｄｓｔｅｐ機構を備えているｄｂｘエンジン。

【請求項７】　前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続する中間インタフェース定義言語（「ＩＤＬ」）生成コード機構のどれを無視すべきかを決定するｒｅｍｏｔｅ　ｓｕｒｒｏｇａｔｅ　ｃｏｄｅ　ｔｅｓｔ機構をさらに含んでいることを特徴とする、請求項６に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用するｄｂｘエンジン。

【請求項８】　指定されたオブジェクトのホストＩＤおよびプロセスＩＤ（ＰＩＤ）を探し出すＧｅｔＩｍｐｌｅｍｅｎｔａｔｉｏｎ機構をさらに含んでいることを特徴とする、請求項６に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用するｄｂｘエンジン。

【請求項９】　遠隔ｄｂｘエンジンが作動しているかどうかを判定し、作動していない場合には、ｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙオブジェクトの機能を使用してｄｂｘエンジンを作成し、遠隔ターゲット機能に接続するＩｄｅｎｔｉｆｙＲｅｍｏｔｅＦｕｎｃｔｉｏｎ機構をさらに含んでいることを特徴とする、請求項６に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用するｄｂｘエンジン。

【請求項１０】　ｄｂｘエンジンが互いに通信を行えるようにする多重ｄｂｘエンジン同期機構をさらに含んでいることを特徴とする、請求項６に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用するｄｂｘエンジン。

【請求項１１】　ローカル・ホスト・コンピュータおよび１つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用するｄｂｘエンジンにおいて、

前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続するが、前記ターゲット・ア

プリケーション・システム自体の一部ではない中間インタフェース定義言語（「ＩＤＬ」）生成コード機構を無視する（すなわち、「ステップ・オーバーする」）ｄｓｔｅｐ機構と、

前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続する中間インタフェース定義言語（「ＩＤＬ」）生成コード機構のどれを無視すべきかを決定するｒｅｍｏｔｅ　ｓｕｒｒｏｇａｔｅ　ｃｏｄｅ　ｔｅｓｔ機構と、

ローカル・ホスト・コンピュータのユーザが実際には遠隔ホスト・コンピュータで実施されている分散ターゲット・アプリケーション・システムの機能にブレークポイントをセットすることを可能とする遠隔ブレークポイント設定機構と、

指定されたオブジェクトのホストＩＤおよびプロセスＩＤ（ＰＩＤ）を探し出すＧｅｔＩｍｐｌｅｍｅｎｔａｔｉｏｎ機構と、

遠隔ｄｂｘエンジンが作動しているかどうかを判定し、作動していない場合には、ｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙオブジェクトの機能を使用してｄｂｘエンジンを作成し、遠隔ターゲット機能に接続するＩｄｅｎｔｉｆｙＲｅｍｏｔｅＦｕｎｃｔｉｏｎ機構と、

ｄｂｘエンジンが互いに通信を行えるようにする多重ｄｂｘエンジン同期機構とを備えているｄｂｘエンジン。

【請求項１２】　ローカル・ホスト・コンピュータおよび１つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用する標準ｄｂｘエンジンを修正するコンピュータ実行方法において、

ＳＰＡＲＣｗｏｒｋｓ　ｄｂｘエンジンのものと同等な機能を有する前記標準ｄｂｘエンジンを設けるステップと、

コンピュータの制御の下で、前記標準ｄｂｘエンジンに、前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続するが、前記ターゲット・アプリケーション・システム自体の一部ではない中間インタフェース定義言語（「ＩＤＬ」）生成コード機構を無視する（すなわち、「ステップ・オーバーする」）ｄｓｔｅｐ機構を設けるステップとを備えているコンピュータ実行方法。

【請求項１３】　コンピュータの制御の下で前記ｄｂｘエンジンに、前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続する中間インタフェース定義言語（「ＩＤＬ」）生成コード機構のどれを無視すべきかを決定するｒｅｍｏｔｅ　ｓｕｒｒｏｇａｔｅ　ｃｏｄｅ　ｔｅｓｔ機構を追加する付加的なステップをさらに含んでいることを特徴とする、請求項１２に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用する標準ｄｂｘエンジンを修正するコンピュータ実行方法。

【請求項１４】　コンピュータの制御の下で前記ｄｂｘエンジンに、ローカル・ホスト・コンピュータのユーザが実際には遠隔ホスト・コンピュータで実施されている分散ターゲット・アプリケーション・システムの機能にブレークポイントをセットすることを可能とする遠隔ブレークポイント設定機構を追加する付加的なステップをさらに含んでいることを特徴とする、請求項１２に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用する標準ｄｂｘエンジンを修正するコンピュータ実行方法。

【請求項１５】　コンピュータの制御の下で前記ｄｂｘエンジンに、指定されたオブジェクトのホストＩＤおよびプロセスＩＤ（ＰＩＤ）を探し出すＧｅｔＩｍｐｌｅｍｅｎｔａｔｉｏｎ機構を追加する付加的なステップをさらに含んでいることを特徴とする、請求項１２に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用する標準ｄｂｘエンジンを修正するコンピュータ実行方法。

【請求項１６】　コンピュータの制御の下で前記ｄｂｘエンジンに、遠隔ｄｂｘエンジンが作動しているかどうかを判定し、作動していない場合には、ｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙオブジェクトの機能を使用してｄｂｘエンジンを作成し、遠隔ターゲット機能に接続するＩｄｅｎｔｉｆｙＲｅｍｏｔｅＦｕｎｃｔｉｏｎ機構を追加する付加的なステップをさらに含んでいることを特徴とする、請求項１２に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用する標準ｄｂｘエンジンを修正するコンピュータ実行方法。

【請求項１７】　コンピュータの制御の下で前記ｄｂｘエンジンに、ｄｂｘエンジンが互いに通信を行えるようにする多重ｄｂｘエンジン同期機構を追加する付加的なステップをさらに含んでいることを特徴とする、請求項１２に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用する標準ｄｂｘエンジンを修正するコンピュータ実行方法。

【請求項１８】　ローカル・ホスト・コンピュータおよび１つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用する標準ｄｂｘエンジンを修正するコンピュータ実行方法において、

ＳＰＡＲＣｗｏｒｋｓ　ｄｂｘエンジンのものと同等な機能を有する前記標準ｄｂｘエンジンを設けるステップと、

コンピュータの制御の下で、前記標準ｄｂｘエンジンに、前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続するが、前記ターゲット・アプリケーション・システム自体の一部ではない中間インタフェース定義言語（「ＩＤＬ」）生成コード機構を無視する（すなわち、「ステップ・オーバーする」）

ｄｓｔｅｐ機構を設けるステップと、
前記標準ｄｂｘエンジンに、前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続する中間インタフェース定義言語（「ＩＤＬ」）生成コード機構のどれを無視すべきかを決定するｒｅｍｏｔｅ　ｓｕｒｒｏｇａｔｅ　ｃｏｄｅ　ｔｅｓｔ機構を設けるステップと、
前記標準ｄｂｘエンジンに、ローカル・ホスト・コンピュータのユーザが実際には遠隔ホスト・コンピュータで実施されている分散ターゲット・アプリケーション・システムの機能にブレークポイントをセットすることを可能とする遠隔ブレークポイント設定機構を設けるステップと、
前記標準ｄｂｘエンジンに、指定されたオブジェクトのホストＩＤおよびプロセスＩＤ（ＰＩＤ）を探し出すＧｅｔＩｍｐｌｅｍｅｎｔａｔｉｏｎ機構を設けるステップと、
前記標準ｄｂｘエンジンに、遠隔ｄｂｘエンジンが作動しているかどうかを判定し、作動していない場合には、ｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙオブジェクトの機能を使用してｄｂｘエンジンを作成し、遠隔ターゲット機能に接続するＩｄｅｎｔｉｆｙＲｅｍｏｔｅＦｕｎｃｔｉｏｎ機構を設けるステップと、
前記標準ｄｂｘエンジンに、ｄｂｘエンジンが互いに通信を行えるようにする多重ｄｂｘエンジン同期機構を設けるステップとを備えているコンピュータ実行方法。
【請求項１９】　ローカル・ホスト・コンピュータおよび１つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムを作成するコンピュータ実行方法において、
ローカル・ホスト・コンピュータに、デバッガＧＵＩおよび１つまたは複数のデバッガ・エンジン（以下、「ｄｂｘエンジン」と呼ぶ）であって、前記デバッガＧＵＩが前記ｄｂｘエンジンと通信し、かつ前記デバッガ・システムのユーザと通信するためのインタフェース機構を備えているデバッガＧＵＩおよび１つまたは複数のｄｂｘエンジンを設けるステップと、
メッセージを互いに送受信するために前記ｄｂｘエンジンと前記デバッガＧＵＩが使用する通信機構を設けるステップと、
前記の１つまたは複数のｄｂｘエンジンの１つであり、前記ローカル・ホスト・コンピュータから離隔したホスト・コンピュータにあり、前記通信機構によって前記デバッガＧＵＩに接続されている遠隔ｄｂｘエンジンを設けるステップとを備えており、前記遠隔ｄｂｘエンジンが前記デバッガＧＵＩからの指示の下で、前記遠隔ホスト・コンピュータにある前記分散ターゲット・アプリケーション・システムの部分に接続して、前記遠隔ホスト・コンピュータにある前記分散ターゲット・アプリケーション・システムの前記部分をデバッグする分散デバッガ・システムを作成するコンピュータ実行方法。
【請求項２０】　ローカル・ホスト・コンピュータおよび１つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムを作成するコンピュータ実行方法において、
デバッガＧＵＩおよび１つまたは複数のデバッガ・エンジン（以下、「ｄｂｘエンジン」と呼ぶ）であって、前記デバッガＧＵＩが前記ｄｂｘエンジンと通信し、かつ前記デバッガ・システムのユーザと通信するためのインタフェース機構を備えており、前記ｄｂｘエンジンが前記ローカルおよび遠隔のコンピュータにあることができるデバッガＧＵＩおよび１つまたは複数のｄｂｘエンジンを設けるステップと、
メッセージを互いに送受信するために前記ｄｂｘエンジンと前記デバッガＧＵＩが使用する通信機構を設けるステップと、
遠隔ホスト・コンピュータにある前記ターゲット・アプリケーション・システムの一部をデバッグする際に使用される新しいｄｂｘエンジンを前記遠隔ホスト・コンピュータに作成するために前記デバッガＧＵＩが使用するｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙ機構を設けるステップとを備えている分散デバッガ・システムを作成するコンピュータ実行方法。
【請求項２１】前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続するが、前記ターゲット・アプリケーション・システム自体の一部ではない中間インタフェース定義言語（「ＩＤＬ」）生成コード機構を無視する（すなわち、「ステップ・オーバーする」）ｄｓｔｅｐ機構を設けるステップをさらに含んでいることを特徴とする、請求項２０に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムを作成するコンピュータ実行方法。
【請求項２２】　前記ｄｂｘエンジンの１つが前記ｄｂｘエンジンの他のものと、これらのｄｂｘエンジンが異なるホスト・コンピュータにあるかどうかにかかわりなく、通信することを可能とする第１の通信機構を設けるステップをさらに含んでいることを特徴とする、請求項２０に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムを作成するコンピュータ実行方法。
【請求項２３】　前記デバッガＧＵＩが前記ｄｂｘエンジンの１つに、前記ｄｂｘエンジンがどのホスト・コンピュータにあるかにかかわりなく、フォーカスすることを可能とする第２の通信機構を設けるステップをさらに含んでいることを特徴とする、請求項２０に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムを作成するコンピュータ実行

方法。

【請求項２４】　前記ユーザが、前記ｄｂｘエンジンがどのホスト・コンピュータにあるかにかかわりなく、すべての活動ｄｂｘエンジンのリストを前記デバッガＧＵＩから取得できるようにする第３の通信機構を設けるステップをさらに含んでいることを特徴とする、請求項２０に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムを作成するコンピュータ実行方法。

【請求項２５】　プロセスを作成し、分散コンピューティング環境において前記プロセスと通信する分散コンピューティング環境におけるコンピュータ実行方法において、

ａ．クライアント・プロセスが要求されているサービスが遠隔であるかどうかを判定するステップと、

ｂ．前記サービスが遠隔であると前記クライアント・プロセスが判定した場合に、前記サービスがおかれている遠隔ロケーションを決定するステップと、

ｃ．前記クライアント・プロセスが前記遠隔ロケーションにおいて前記サービスを有しているサーバ・プロセスにメッセージを送って、前記サーバに前記クライアント・プロセスとの通信を確立するよう指示するステップと、

ｄ．前記サーバ・プロセスが前記クライアント・プロセスと通信するインタフェース・プロセスを作成するステップと、

ｅ．前記インタフェース・プロセスが前記サーバ・プロセスに接続するまで、前記サーバ・プロセスが操作を中断するステップと、

ｆ．前記インタフェース・プロセスが前記サーバ・プロセスへの接続を試みるステップと、

ｇ．前記インタフェース・プロセスが前記サーバ・プロセスに接続できない場合に、前記インタフェース・プロセスが前記クライアント・プロセスにアラートし、そうでない場合には、前記クライアント・プロセスが前記サーバ・プロセスに接続して、前記接続が成功した旨を前記クライアント・プロセスにアラートするステップと、

ｈ．前記クライアント・プロセスが前記インタフェース・プロセスを介して前記サーバ・プロセスと通信するステップと

を備えている分散コンピューティング環境におけるコンピュータ実行方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は分散コンピューティング・システム、クライアント・サーバ・コンピューティングおよびオブジェクト指向プログラミングの分野に関する。具体的にいえば、本発明はプログラム開発者およびユーザに分散サーバ上にプログラムまたはオブジェクトを含んでいることのあるターゲット・アプリケーションをデバッグする能力を与える方法および装置に関する。

【０００２】

【従来の技術】コンピュータ・アプリケーション開発者が作成中のアプリケーションをデバッグできることは必須である。これはオブジェクト指向分散プロセッサ環境においては、きわめて困難な問題となっている。このような現代の環境はアプリケーション開発者以外の人間が開発したオブジェクトを呼び出すアプリケーションを含んでおり、かつアプリケーション開発者から遠隔の、当該開発者には未知のプロセッサで作動するオブジェクトのインプリメンテーションを含んでいる可能性もある。それにもかかわらず、アプリケーション開発者には、このような遠隔で、未知のプロセッサに常駐するアプリケーションの部分をデバッグする方法がなければならない。

【０００３】たとえば、分散アプリケーションは２つ以上の部分からなるアプリケーションである。これらの部分はクライアントおよびそのサーバと呼ばれることがしばしばある。分散アプリケーションは何年も前から存在しているものであり、それだけの年月の間、プログラム・アプリケーション開発者は分散アプリケーションをデバッグする問題を抱えていた。分散アプリケーションの典型的なデバッグ法は、デバッガの下でクライアントを起動し、サーバにある機能に達するまでクライアントをデバッガすることである。開発者が幸運であれば、サーバがすでに既知のホストで作動している。開発者は次いでサーバ・ホストへ進み、サーバ・プロセスを識別し、デバッガをこれに接続し、デバッグ・セッションを継続する。サーバがまだ作動していない場合には、開発者はサーバを作動させる方法を見つけだし、自分が行ったことで探しているバグが隠ぺいされないよう祈る他はなくなる。サーバが起動されたら、開発者はデバッガをこれに接続する。あるいは、サーバの起動に干渉して、何か興味深いことが起きる前にデバッガをサーバに接続する方法を、開発者が見つけださなければならない。この方法はエラーを起こしやすいものであり、手間がかかり、またしばしばわかりにくく、単調なものである。

【０００４】オブジェクト指向システムにおいて、オブジェクトはデータと、データを処理するために呼び出すことのできる操作（オペレーション）とからなる構成要素である。操作（「メソッド」とも呼ばれる）はオブジェクトにコールを送ることによってオブジェクトに呼び出される。各オブジェクトはオブジェクト・タイプを有しており、このタイプはそのタイプのオブジェクトで行うことのできる操作を定義している。あるオブジェクト・タイプは他のオブジェクト・タイプについて定義され、実施されたオブジェクト操作を継承することができる。オブジェクト指向設計およびプログラミング技法の詳細については、Ｂｅｒｔｒａｎｄ　Ｍｅｙｅｒの「Ｏ

ｂｊｅｃｔ－ｏｒｉｅｎｔｅｄ　Ｓｏｆｔｗａｒｅ　Ｃｏｎｓｔｒｕｃｔｉｏｎ」、Ｐｒｅｎｔｉｃｅ－Ｈａｌ、１９８８年を参照されたい。

【０００５】クライアント・サーバ・コンピューティングにおいては、通常、コンピュータを接続しているネットワークを介して互いに通信を行える１組のコンピュータがある。これらのコンピュータの中には、他のコンピュータに対するサービスまたは機能のプロバイダとして働くものがある。サービスまたは機能のプロバイダは「サーバ」と呼ばれ、サービスまたは機能の消費者は「クライアント」と呼ばれる。クライアント・サーバ・モデルは、同一のコンピュータで作動している別個のプログラムが何らかの保護機構を介して互いに通信を行い、機能のプロバイダと消費者として働いているというケースに一般化することもできる。

【０００６】クライアント・サーバ・モデルに基づくオブジェクト指向分散システムにおいては、オブジェクト指向インタフェースをクライアントに与えるサーバがある。これらのサーバは、データと、このタイプのオブジェクトで認められる操作にしたがってデータを処理する関連ソフトウェアとからなるオブジェクトを、サポートしている。クライアントはこれらのオブジェクトに対するアクセスを取得し、コールをサーバに伝送することによってこれらのオブジェクトでコールを実行することができる。サーバ側では、これらのコールはオブジェクトに関連するソフトウェアによって実行される。これらのコールの結果はクライアントへ返送される。

【０００７】従来技術のデバッガによる他の基本的な問題は、現代の分散オブジェクト・システムで実施されたアプリケーションをデバッグしようとするときに生じる。オブジェクト管理グループ（「ＯＭＧ」）によって一般的に規定されたタイプの分散オブジェクト・システムを考える。ＯＭＧは分散オブジェクト・システムのいくつかの仕様およびプロトコルに合意した５００社以上の会社の団体である。このシステムの基本仕様は「Ｔｈｅ　Ｃｏｍｍｏｎ　Ｏｂｊｅｃｔ　Ｒｅｑｕｅｓｔ　Ｂｒｏｋｅｒ：Ａｒｃｈｉｔｅｃｔｕｒｅ　ａｎｄ　Ｓｐｅｃｉｆｉｃａｔｉｏｎ」（「ＣＯＲＢＡ」ともいう）という題名の１９９３年１２月２９日付のＯＭＧ　Ｄｏｃｕｍｅｎｔ　Ｎｏ．９３．ｘｘ．ｙｙ　Ｒｅｖｉｓｉｏｎ　１．２に記載されている（この基本仕様はここでの引用によって本明細書の一部となる）。ＣＯＲＢＡ準拠システムは既存のオブジェクトによってアプリケーションを構築することに備えたものである。このようなアプリケーションはオブジェクトの作成を要求し、かつそのオブジェクトで操作を行うことができる。オブジェクトの作成およびその上での操作はこれらのオブジェクトのサーバによって行われる。このようなアプリケーションはオブジェクトを作成しようとした場合、そのオブジェクトの「ファクトリ」と呼ばれるサーバを探し出すロケータ機構を透過的に利用する。同様に、このようなアプリケーションが既存のオブジェクトを有している場合には、ロケータ機構を透過的に利用して、そのオブジェクトでの操作を行うことのできるサーバを探し出す。

【０００８】

【発明が解決しようとする課題】このようなＣＯＲＢＡ準拠システムにおいては、アプリケーション・プログラマがそのオブジェクトのサーバが作動している場所についての知識なしにオブジェクトを使用するのを可能とする相当な量の機構（メカニズム）が各オブジェクトの背後にある。クライアントの開発者がサーバの開発者でもあるという特別な状況では、サーバがどこで作動することになるのか、またその名前が何かということをプログラマが知るような構成をとることができる。しかしながら、一般に、ＣＯＲＢＡ準拠システムのアプリケーション開発者は自分のオブジェクトに関連したサーバを探し出すことができない。それ故、オブジェクトが同一のプロセスに配置されているのか、遠隔のプロセスに配置されているのかにかかわりなく、またオブジェクトがどこにあるのかにかかわりなく、アプリケーションが使用するオブジェクトのデバッグをサポートする必要がある。さらに、このデバッグ手順が開発者に「単一プロセス」であると思わせ、熟知したデバッグ・パラダイムを使用して大規模な分散アプリケーションのデバッグを行えるようにすることが好ましい。

【０００９】いくつかの従来技術のデバッガの主な欠点は、これらがデータおよびプロセスをサポートするのに、通常は関連するコンパイラが生成する付加的なデータを含んでいる大きなオーバヘッドを必要とすることである。したがって、遠隔デバッガの好ましい実施の形態は、オブジェクトの実装者がオブジェクトを「デバッグ可能」にするのに、サーバをコンパイルして、記号情報を生成する以外の特別なことを何もする必要のないようにする必要がある（－ｇコンパイラ・オプションを使用したＣおよびＣ＋＋の場合）。しかしながら、関連するオーバヘッドが細分化されたオブジェクトのサイズおよびパフォーマンスに影響を及ぼさないように、サーバまたはサーバントのいずれかでの付加的な挙動上の抽象化やデータの抽象化を必要としてはならない。同様に、従来技術のデバッギング・システムの他の制限は、これらが特定のタイプのターゲット・アプリケーションおよび／または特定のコンパイラ言語にリンクされていることである。分散デバッガが実装言語と無関係にアプリケーションをデバッグできることが望ましい。すなわち、好ましい分散デバッガはこれが作動するサーバおよびオブジェクトの種類についていかなる想定もしてはならない。ＣＯＲＢＡ仕様は各種のオブジェクトの実装（インプリメンテーション）をサービスするのに使用される各種の「オブジェクト・アダプタ」を記述している。望ましい分散デバッガは、これが作動するサーバやオブジェ

クトの種類について実際にいかなる想定もしないのであれば、「オブジェクト・アダプタ」に無関係な態様で作動しなければならない。さらに、ＣＯＲＢＡ準拠システムのインプリメンテーションがシステムの作動を容易とするために使用することのあるボイラプレート・コードを無視できる分散デバッガを得ることが望ましい。「ボイラプレート」コードとは開発環境によって作成され、開発者にはわからない非プログラマ生成コードをいう。分散デバッガは開発者がシステムを実装した抽象化と同じ機能レベルでシステムをデバッグすることを可能とするものでなければならない。

【００１０】「ｄｏｅＤｅｂｕｇｇｅｒ」と呼ぶ本発明の分散デバッガはシームレスで、オーバヘッドが少なく、わずらわしくない態様で分散デバッギングを達成し、これによって開発者が「単一プロセス」のアプリケーションをデバッグしているのだと思いこんで、分散オブジェクト指向アプリケーションをデバッグすることを可能とする装置および方法を提供する。

【００１１】

【課題を解決するための手段】クライアント・アプリケーションがローカル・ホストでデバッガを使用できるとともに、オブジェクトを含んでいるアプリケーションおよび未知の遠隔ホスト・コンピュータで作動するオブジェクト・インプリメンテーションをシームレスにデバッグできる装置および方法を開示する。

【００１２】一部がローカル・ホスト・コンピュータにあり、一部が１つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムにおいて、ローカル・ホスト・コンピュータまたは遠隔ホスト・コンピュータにおかれているデバッガＧＵＩおよび１つまたは複数のｄｂｘエンジンと、ｄｂｘエンジンとデバッガＧＵＩが互いに会話を行うために使用する通信機構とを有している分散デバッガ・システムを開示する。

【００１３】特許請求の範囲に記載する本発明の他の態様は、一部がローカル・ホスト・コンピュータにあり、一部が遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムにおいて、ローカル・ホスト・コンピュータまたは遠隔ホスト・コンピュータにおかれているデバッガＧＵＩおよび１つまたは複数のｄｂｘエンジンと、ｄｂｘエンジンとデバッガＧＵＩが互いに会話を行うために使用する通信機構とを有しており、さらに希望する遠隔デバッギング・サポートをもたらすため、必要に応じ、遠隔ホスト・コンピュータに新しいｄｂｘエンジンを作成するのにデバッガＧＵＩが使用するｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙ機構を有している分散デバッガ・システムを含む。

【００１４】本発明はこれらの操作を行う装置にも関する。この装置は必要とされる目的に合わせて特に構成されたものであっても、コンピュータに記憶されているコンピュータ・プログラムによって選択的に起動されるか、あるいは再構成される汎用コンピュータを備えたものであってもよい。本明細書に記載する作動は特定のコンピュータまたはその他の装置に本質的に関連したものではない。詳細にいえば、各種の汎用マシンを本発明の教示にしたがって作成されたプログラムとともに使用してもよいし、あるいは必要とされる方法のステップを行うための専用マシンの方が便利なこともある。各種のこれらのマシンに必要な構造は以下の説明から明かとなろう。

【００１５】特許請求の範囲に記載する本発明の他の態様は、ローカル・ホスト・コンピュータと１つまたは複数の遠隔ホスト・コンピュータ装置におかれる分散・ターゲット・アプリケーション・システムのデバッグ時に分散デバッガ・システムで使用するｄｂｘエンジンにおいて、非プログラマ生成コードを無視するｄｓｔｅｐ機構と、このような非プログラマ生成コード（ＩＤＬ生成コードともいう）を識別するテスト機構と、ターゲット・アプリケーション・システムのセクションに遠隔ブレークポイントをセットする機構と、呼び出されたオブジェクトを実施するサーバのホストＩＤおよびプロセスＩＤ（ｐｉｄ）を識別するＧｅｔＩｍｐｌｅｍｅｎｔａｔｉｏｎ機構（「サーバ検索」機構ともいう）と、ｄｂｘエンジンが互いに通信を行えるようにする多重ｄｂｘエンジン同期機構とからなるｄｂｘエンジンを含んでいる。

【００１６】本願の特許請求の範囲には、上記の特性を有するｄｂｘエンジンを作成する方法、ならびに上記のような分散デバッガ・システムを作成する方法も記載する。

【００１７】本発明の目的、特徴および利点は以下の説明から明かとなろう。

【００１８】

【発明の実施の形態】表記および用語以下の詳細な説明はコンピュータまたはコンピュータのネットワークで実行されるプログラム・プロシージャによって表される。これらのプロシージャの記述および表現は当分野の技術者が自分達の作業の主題を他の技術者に最も効率よく伝えるために使用する手段である。

【００１９】プロシージャは本明細書において、また一般に、希望する結果にいたる一貫した一連のステップであると考えられる。これらのステップは物理量の物理的取扱いを必要とするものである。必ずしもそうとは限らないが、一般に、これらの量は記憶、転送、組合せ、比較、あるいは処理を行える電気信号または磁気信号の形態をしている。主として一般的に使用するため、これらの信号をビット、値、要素、記号、文字、項、数値などと呼ぶのが便利なこともある。ただし、これらや類似した用語が適切な物理量と関連しており、これらの量に適

用された都合のよいラベルに過ぎないことに留意すべきである。

【００２０】さらに、実行される処理は操作員が行う知的な操作と一般に関連づけられている乗算や比較などの用語で呼ばれることがしばしばある。本発明の一部を形成する本明細書に記載する操作のいずれにおいても、操作員のこのような能力は必要ではなく、またほとんどの場合、望ましくない。操作はマシン操作である。本発明の操作を行うのに有用なマシンとしては、汎用ディジタル・コンピュータまたは類似の装置がある。

【００２１】本発明はこれらの操作を行う装置にも関する。この装置は必要とされる目的に合わせて特に構成されたものであっても、コンピュータに記憶されているコンピュータ・プログラムによって選択的に活動化されるか、あるいは再構成される汎用コンピュータを備えたものであってもよい。本明細書に記載する作動は特定のコンピュータまたはその他の装置に本質的に関連したものではない。各種の汎用マシンを本発明の教示にしたがって作成されたプログラムとともに使用してもよいし、あるいは必要とされる方法のステップを行うための専用マシンの方が便利なこともある。各種のこれらのマシンに必要な構造は以下の説明から明かとなろう。

【００２２】以下の開示はアプリケーションのプログラマ／開発者が１台のホストマシンのところにおり、開発されるアプリケーションがプログラマ／開発者には未知の他のホストマシンにおかれていることのあるオブジェクトおよびオブジェクト・インプリメンテーションを利用する分散コンピュータ・アプリケーションをデバッグするシステムおよび方法に関する。このシステムおよび方法は広く分散したオブジェクト指向クライアント・サーバ・システムにおけるオブジェクトの使用に関連づけられた新しいアプリケーションをデバッグを試みる際に遭遇する問題の解決策を提供する。分散オブジェクトはこれらが要求を他のオブジェクトへ送るのか、あるいはクライアントからの要求に応答しているのかに応じてオブジェクト・クライアントにも、オブジェクト・サーバにもなり得る。分散オブジェクト環境において、要求および応答はオブジェクトの場所および状況を認識しているオブジェクト・リクエスト・ブローカ（ＯＲＢ）によって行われる。ＯＲＢを実施するのに適している１つのアーキテクチャが、共通オブジェクト・リクエスト・ブローカ・アーキテクチャ（ＣＯＲＢＡ）仕様によって与えられる。任意の関連した状況で使用できるものであるが、記載されているインプリメンテーションはＳｕｎ Ｍｉｃｒｏｓｙｓｔｅｍｓ，Ｉｎｃ．の分散オブジェクト環境（「ＤＯＥ」）の拡張である。しかしながら、本開示で記載するプロセスおよびシステムを理解し、実施するのに、当分野の技術者がＤＯＥシステムの詳細な知識を必要とすることはない。

【００２３】本発明はプログラマ／開発者のアプリケーションが呼び出したオブジェクトが遠隔で実施されていることを、プログラマ／開発者が無視できるようにし、また遠隔インプリメンテーションであることによる特別なことを、プログラマ／開発者が行うことを要求せず、しかも過度のオーバヘッド負荷を課すことがなく、安全であり、プログラマ／開発者がシステムを実施したのと同じ抽象化の機能レベルで、プログラマ／開発者がシステムをデバッグするのを可能とするｄｏｅＤｅｂｕｇｇｅｒを作成し、使用するシステムおよび方法を開示する。他のインプリメンテーションは若干の言語非依存性ももたらす。

【００２４】Ｉ．定義

本明細書では、「分散オブジェクト」または「オブジェクト」という用語は、オブジェクトと関連づけられた定義済みのインタフェースを介した操作によって処理できるコードおよびデータのカプセル化されたパッケージをいう。それ故、当分野の技術者にとって、分散オブジェクトは従来のプログラミング・オブジェクトを定義している基本特性を含むものとみなされる。しかしながら、分散オブジェクトは従来のプログラミング・オブジェクトとは、２つの重要な特徴を含んでいることによって相違している。まず、分散オブジェクトは多言語のものである。分散オブジェクトのインタフェースは各種の異なるプログラミング言語にマップできるインタフェース定義言語を使用して定義される。このようなインタフェース定義言語の１つがＯＭＧ　ＩＤＬである。第２に、分散オブジェクトは場所に依存しない。すなわち、分散オブジェクトはネットワーク内のどこにでもおける。これは単一のアドレス・スペース、すなわち「クライアント」のアドレス・スペースに存在しているのが典型的な従来のプログラミング・オブジェクトと明確に異なるところである。分散オブジェクトは他のオブジェクトに要求を送っているのか、あるいは他のオブジェクトからの要求に応答しているのかに応じてオブジェクト・クライアントにも、オブジェクト・サーバにもなり得る。要求および応答はオブジェクトの場所および状況を認識しているオブジェクト・リクエスト・ブローカ（ＯＲＢ）によって行われる。

【００２５】「分散オブジェクト・システム」ないし「分散オブジェクト・オペレーティング環境」とは、ＯＲＢを介して通信を行う分散オブジェクトからなるシステムをいう。

【００２６】「オブジェクト・リファレンス」ないし「ｏｂｊｒｅｆ」とは、他のオブジェクトへのポインタを含んでいるデータ構造（従来のプログラミング言語オブジェクトでもよい）をいう。オブジェクト・リファレンスの作成および定義については、当分野の技術者は熟知しているであろう。

【００２７】本明細書で定義する「クライアント」とは、オブジェクトに要求を送るエンティティである。こ

のモデルにおいて、オブジェクトは、サービスを与えるものであり、「サーバ・オブジェクト」または「ターゲット・オブジェクト」または「オブジェクト・インプリメンテーション」と呼ばれる。したがって、クライアントはサーバから操作、ないしインプリメンテーションを呼び出す。場合によっては、クライアントはそれ自体でオブジェクトである。分散オブジェクト環境において、クライアントはインプリメンテーション・プログラミング言語についての知識を持っていなくてもよく、あるいは、このようなオブジェクトの多言語性の要件により、インプリメンテーションがクライアントのプログラミング言語の知識を持っていなくてもよい。分散オブジェクト・オペレーティング環境のクライアントおよびサーバに必要なのは、インタフェース定義言語によって通信を行うことだけである。上述したように、クライアントによるサーバに対する要求およびクライアントに対するサーバの応答は、ＯＲＢによって処理される。指摘しておきたいのは、クライアントとサーバが同一のホスト・コンピュータまたは２台の異なるホスト・コンピュータの同一のプロセスに存在できることである。

【００２８】「オブジェクト・インタフェース」はオブジェクトがもたらす操作（オペレーション）、属性（アトリビュート）および例外事項（エクセプション）の仕様である。分散オブジェクトのオブジェクト・インタフェースは許可されているインタフェース定義言語（ＩＤＬ）を使用して作成することが好ましい。上記したように、オブジェクトはそのインタフェースを介してトランザクションを行う。したがって、インタフェースを使用することによって、トランザクションにおけるオブジェクトのメソッドおよびデータを定義するのに使用されたプログラミング言語をクライアントが認識している必要がなくなる。

【００２９】情報のパケットを「マーシャル（ｍａｒｓｈａｌ）」するとは、共用メモリ通信チャネル、あるいはネットワーク通信回線のいずれかによってこの情報を転送する準備を行うことである。これはしばしば、使用されている通信プロトコルにしたがって特定のフォーマットでデータを編成することをいう。

【００３０】情報のパケットを「アンマーシャル（ｕｎｍａｒｓｈａｌ）」するとは、本質的にマーシャル手順を逆にすることであり、該当する環境で有意のフォーマットでデータを作成することである。

【００３１】「ＴｏｏｌＴａｌｋ」とは、本願出願人Ｓｕｎ　Ｍｉｃｒｏｓｙｓｔｅｍｓ，Ｉｎｃ．の子会社であるＳｕｎＳｏｆｔが開発し、提供している通信システムである。「ＴｏｏｌＴａｌｋ」はあるアプリケーションが作成したメッセージを、そのメッセージを受け取ることを要求している他のものへ送出する通信サービスを提供する。「ＴｏｏｌＴａｌｋ」によって、独立したアプリケーションが他のアプリケーションと、相互の直接的な知識を要することなく、通信を行うことが可能となる。送信者は「ＴｏｏｌＴａｌｋ」メッセージを特定のプロセス、何らかの関心のあるプロセス、オブジェクト、あるいはオブジェクト・タイプにアドレス指定することができる。「ＴｏｏｌＴａｌｋ」の詳細については「ＴｈｅＴｏｏｌＴａｌｋ　Ｓｅｒｖｉｃｅ：Ａｎ　Ｉｎｔｅｒ－Ｏｐｅｒａｂｉｌｉｔｙ　Ｓｏｌｕｔｉｏｎ，（ＩＳＢＮ０１３－０８８７１７－Ｘ）という題名のＳｕｎＳｏｆｔ　Ｐｒｅｓｓ／ＰＴＲ　Ｐｒｅｎｔｉｃｅ　Ｈａｌｌの書籍に記載されている。

【００３２】ＩＩ．作動環境

本発明が使用される環境には、汎用コンピュータ、ワークステーションあるいはパーソナル・コンピュータがさまざまなタイプの通信リンクを介して、クライアント・サーバ構成で接続されており、プログラムやデータが多くがオブジェクトの形式で、システムの各種のメンバによって、システムの他のメンバによる実効およびアクセスのために利用できる汎用分散コンピューティング・システムが含まれている。汎用ワークステーション・コンピュータの要素のいくつかを図１に示す。図において、入出力（「Ｉ／Ｏ」）部２、中央処理装置（「ＣＰＵ」）３およびメモリ部４を有するコンピュータ１が示されている。Ｉ／Ｏ部２はキーボード５、表示装置６、ディスク記憶装置９、およびＣＤ－ＲＯＭ駆動装置７に接続されている。ＣＤ－ＲＯＭ装置７は、通常、プログラム１０やデータを含んでいるＣＤ－ＲＯＭ媒体８を読み取ることができる。図２は典型的なマルチプロセッサ分散コンピュータ・システムを示しており、個々のコンピュータ２０、２２および２４が通信リンク２６を介して互いに接続され、かつ恐らくは共用メモリ装置２８に接続されている。図３は典型的なオブジェクト指向クライアント・サーバ構成を示しており、ユーザ３０が第１のコンピュータ３２でクライアント・アプリケーション３４を起動できる。クライアント・アプリケーション３４はコール４０をオブジェクト・リファレンス３６に出し、このオブジェクト・リファレンスは第２のコンピュータ（サーバ）５０のオブジェクトのインプリメンテーション（「ターゲット・オブジェクト」ともいう）４６をポイントする。コール４０は通信制御機構３８に渡され、この機構３８はコールをオブジェクト・インプリメンテーション４６が配置されているサーバ５０に送る。このオブジェクト・インプリメンテーション４６は最初にオブジェクト・リファレンス３６を作成し、ユーザに利用できるようにする。コールの処理の終了時に、オブジェクト・インプリメンテーション４６はメッセージまたは希望した操作の結果を通信リンク４２を介して起点クライアント・アプリケーション３４に返す。このクライアント・サーバ・モデルは、通信機構の機能がオペレーティング・システム（図４の６２）によって行われる単一プロセッサ装置でも機能する。

【００３３】ＩＩＩ．分散デバッガー作成方法
個々で、図５を参照すると、ＳＰＡＲＣｗｏｒｋｓデバッガ・システムが示されている。ＳＰＡＲＣｗｏｒｋｓデバッガ（以下、「Ｄｅｂｕｇｇｅｒ」と呼ぶ）はＳｕｎ　Ｍｉｃｒｏｓｙｓｔｅｍｓ，Ｉｎｃ．製のＳＰＡＲＣｗｏｒｋｓツールセットの組込みコンポーネントである。なお、ＳＰＡＲＣｗｏｒｋｓツールセットはＡｎａｌｙｚｅｒ、ｄｂｘエンジン、ＦｉｌｅＭｅｒｇｅツール、Ｍａｋｅｔｏｏｌ、Ｍａｎａｇｅｒ、および、ＳｏｕｒｃｅＢｒｏｗｓｅｒを含む。Ｄｅｂｕｇｇｅｒの詳細については、ＳｕｎＳｏｆｔが１９９４年８月にＰａｒｔ　Ｎｏ．８０１－７１０５－１０として刊行した「Ｄｅｂｕｇｇｉｎｇ　ａＰｒｏｇｒａｍ」という題名の文献に記載されている（これは引用により本明細書の一部とする）。

【００３４】ここで、図５を参照すると、Ｄｅｂｕｇｇｅｒはｄｂｘエンジン７６とインタフェースする洗練されたウィンドウ・ベースのツール７２（Ｄｅｂｕｇｇｅｒ・グラフィカル・ユーザ・インタフェース（ＧＵＩ））を含んでいる。ｄｂｘエンジン７６は対話式のライン指向ソースレベル記号デバッガである。ｄｂｘエンジン７６はターゲット・プログラムがクラッシュした場所を決定し、変数と式の値を調べ、コードにブレークポイント７８をセットし、ターゲット・プログラムを実行し、トレースすることを可能とする。プログラムの実行中に、ｄｂｘエンジン７６はターゲット・プログラムの挙動に関する詳細な情報を取得し、ＴｏｏｌＴａｌｋ通信プロトコル７４によってＤｅｂｕｇｇｅｒ　ＧＵＩ７２にこの情報を供給する。ｄｂｘエンジン７６はコンパイラがコンパイラ・オプション－ｇを使用して生成したデバッギング情報に依存して、ターゲット・プロセスの状態を検査する。現行のＳｕｎ　Ｍｉｃｒｏｓｙｓｔｅｍｓ，Ｉｎｃ．のオペレーティング・システム環境であるＳｏｌａｒｉｓ　２．ｘのデフォルトでは、各プログラム・モジュールのデバッギング情報はモジュールの　．０ファイルに格納される。Ｓｏｌａｒｉｓ　２．ｘにおける好ましい実施の形態において、ｄｂｘエンジン７６は必要に応じ、各モジュールに関する情報を読み込む。「ブレークポイント設定」７８の機能に加えて、ｄｂｘエンジン７６は「単一ステップ」８０の機能、ならびに上記の参照文献「Ｄｅｂｕｇｇｉｎｇ　ａ　Ｐｒｏｇｒａｍ」に詳細が記載されている多くのその他の機能８２を有している。「ステップ」８０機能はプログラマ／開発者がターゲット・プログラム・コードをソース言語レベルまたはマシン語レベルのいずれかで１行ずつ単一のステップで実行し、ファンクションコールをステップ「ｏｖｅｒ」またはステップ「ｉｎｔｏ」し、ファンクション・コールをステップ「ｕｐ」またはステップ「ｏｕｔ」してコール側ファンクション行に到達すること（ただし、コール後に）を可能とする。ブレークポイント７８のコマンドには３つのタイプがある。

【００３５】（１）ｓｔｏｐタイプのブレークポイント：　ターゲット・プログラムがｓｔｏｐコマンドを使用して作成されたブレークポイントに到達した場合、プログラムは停止し、任意選択で１つまたは複数のデバッグ・コマンドを発行する。ターゲット・プログラムをレジュームするには、他のデバッグ・コマンドを発行しなければならない。

【００３６】（２）ｗｈｅｎタイプのブレークポイント：　ターゲット・プログラムが停止し、ｄｂｘエンジンが１つまたは複数のデバッグ・コマンドを発行し、その後ターゲット・プログラムが継続する。

【００３７】（３）ｔｒａｃｅタイプのブレークポイント：　ターゲット・プログラムが停止し、イベント固有のｔｒａｃｅ行が排除され、その後プログラムが継続する。

【００３８】非分散システムにおける、Ｄｅｂｕｇｇｅｒの典型的な構成を図６に示す。図において、ＴｏｏｌＴａｌｋ通信リンク９６によってｄｂｘエンジン９８に接続されたデバッガＧＵＩ９４を含んでいるホストマシン９２が示されており、ｄｂｘエンジン９８はクライアント（ターゲット・プログラム）１００にリンクされており、クライアントはさらに付加的なターゲット・アプリケーション・コード（サーバ）１０２に接続されている。分散システムにおいて、プログラマ／開発者は図７に示したような状態に直面する。図７は複数のホスト上にある複数のクライアントと複数のサーバを示している。赤ホスト１１２にあるＤｅｂｕｇｇｅｒ１１４を使用して、クライアント１　１１６をデバッグするプログラマ／開発者には、クライアント１　１１６が赤ホスト１１２のサーバ１１８または青ホスト１２２のサーバ１２４によって実行できるオブジェクトで操作を行っており、クライアント１　１１６の開発者としてのプログラマにはコールの実行にどのサーバが使用されるのかがわからないということが判明する。さらに、クライアント１　１１６によって使用されるどのサーバも、白ホスト１３０のクライアント２　１２８が使用できる。ＤＯＥ（本発明の好ましい実施の形態）などのＣＯＲＢＡ準拠の分散システムでは、かなりの量の機構（メカニズム）が各オブジェクトの背後にあり、この機構によって、アプリケーション・プログラマがオブジェクトを、そのオブジェクトのサーバがどこで作動しているのかの知識を必要とせずに、使用できるようになる。このことはこのような状況の下では、分散アプリケーションのデバッグ作業の重大な障害となる。さらに、すべてのＤＯＥサーバはマルチスレッド・サーバであり、サーバが異なるクライアントからの複数の要求に同時に応えることができる。したがって、本発明のｄｏｅＤｅｂｕｇｇｅｒはＤＯＥ分散環境で生じるデバッグ上の問題の多くに対する解決策となる。

【００３９】本発明のｄｏｅＤｅｂｕｇｇｅｒを作成するのに必要なＤｅｂｕｇｇｅｒの改変について説明する前に、オブジェクトがＣＯＲＢＡ準拠環境で互いに通信するのを可能とするためにＤＯＥが使用するボイラプレート機構コードについて簡単に触れておくことが有用であろう。図８を参照すると、ユーザ・コードと基礎ＤＯＥ機構の間の分割が示されている。

【００４０】図８において、ローカル・ホスト１４２で、ユーザ・コードが機能、例えば、機能「ｆｏｏ」１４４を呼び出す。ＤＯＥシステムは機能ｆｏｏとのインタフェースをもたらす生成コード１４８を備えており、生成コード１４８はメッセージ機能１５０を呼び出して、適切なメッセージを作成し、これをサーバ１５２に送る。これらのメッセージ１５４は該当するホスト１５６に到達し、サーバ側メッセージ・システム１５８によって受け取られ、このメッセージ・システムはメッセージをＤＯＥシステム１６０が生成したコードであるサーバ側スタブに渡す。ＤＯＥシステム１６０は次いで機能ｆｏｏ１６２のインプリメンテーション・コードを呼び出し、この機能のインプリメンテーションのユーザ・コードがコール１６４を処理する。アプリケーション・プログラマはｄｏｅサーバによってもたらされるオブジェクトを利用するコード１４４を書く。アプリケーション・プログラマはインタフェース定義言語（ＩＤＬ）を使用してオブジェクトに対するインタフェースを定義する。ＩＤＬはクライアント側およびサーバ側のライブラリにコンパイルされ、これらのライブラリはクライアントにあるオブジェクトに対する操作をサーバで行うことを可能とする。クライアント側ライブラリは操作をサーバに対する要求１５４に変換するスタブ機能１４８、１５０からなっている。サーバ側ライブラリ１６０、１６２はクライアントからの要求をその操作を実施するユーザ提供機能１６４に変換する。

【００４１】たとえば、図８において、ユーザは機能ｆｏｏをＩＤＬに定義している。ＩＤＬコンパイラはスタブ機能ｆｏｏ１４８、１５０を生成し、サーバに送られるメッセージを作成させる（１５２）。サーバ側では、ＩＤＬが生成したコード１６０、１６２がクライアントからのメッセージを取り入れ、これをプログラマによって与えられる、ｆｏｏの機能を実現する、機能ｆｏｏに対するコールに変換する（１６４）。この基礎コード１４８、１５０、１５２、１５８、１６０および１６２はすべてプログラマ／開発者がデバッグしようと思っておらず、また正規のデバッグ・セッションにおいてデバッガに調べさせようと思っていないコードである。したがって、分散デバッギング・セッションがプログラマ／開発差に関する限り、この基礎コードをすべて無視できるようにすることが望ましい。

【００４２】図９はＳＰＡＲＣｗｏｒｋｓデバッガを本発明のｄｏｅＤｅｂｕｇｇｅｒに変換するのに必要な改変および拡張を示す。実際の拡張は共用ライブラリ「ｌｉｂｄｏｅＤｅｂｕｇｇｅｒ．ｓｏ」２０４にパッケージされる。必要とされる基本的な拡張には次のようなものがある。

【００４３】「ｄｓｔｅｐ」コマンド２０６
「ｒｅｍｏｔｅ　ｓｕｒｒｏｇａｔｅ　ｃｏｄｅ　ｔｅｓｔ」機構２０８
「Ｒｅｍｏｔｅ　Ｂｒｅａｋｐｏｉｎｔ」設定機構２１０
「ＧｅｔＩｍｐｌｅｍｅｎｔａｔｉｏｎ」機構２１２
「ＩｄｅｎｔｉｆｙＲｅｍｏｔｅＦｕｎｃｔｉｏｎ」機構２１４
「ｍｕｌｔｉｐｌｅ　ｄｂｘ　ｅｎｇｉｎｅ　ｓｙｎｃｈｒｏｎｉｚｅｒ」機構２１６

【００４４】これらの拡張に加えて、ｄｏｅＤｅｂｕｇｇｅｒの作動をサポートするためのＤｅｂｕｇｇｅｒ　ＧＵＩおよびｄｂｘエンジンの改変には、次のものがある。

【００４５】あるｄｂｘエンジンが他のｄｂｘエンジンと通信を行うための通信機能
デバッガーＧＵＩからすべての活動ｄｂｘエンジンのリストを取得する機能

【００４６】これらの改変の好ましい実施の形態を総括的に説明するが、当分野の技術者にはこれらの機能および特徴が最も適切なものである部分に対するハードウェア機構および装置を含む多くの形態で実現できることが認識されよう。

【００４７】「ｄｓｔｅｐ」コマンドは分散システムのどこに所与の機能のインプリメンテーションが実際にあるのかにかかわりなく、所与の機能のインプリメンテーションにシームレスに取りかかるために、プログラマ／開発者によって使用される。「ｄｓｔｅｐ」はまず正規のｄｂｘ「ｓｔｅｐ」コマンドを発行することによって作動する。標準の「ｓｔｅｐ」コマンドはデバッグされるプロセス（デバッギー）の実行をデバッギーの現行のソース・ラインから次のソース・ラインまで継続する。現行ラインが機能を呼び出す点になると、次のソース・ラインが呼び出される機能の最初のソース・ラインになる。ｓｔｅｐコマンドのセマンティックスを分散アプリケーションのデバッギングに拡張するため、拡張されたｓｔｅｐが図８のクライアントのｆｏｏなどの機能に入ると（点Ａ　１４６）、実行がサーバのｆｏｏのインプリメンテーションの最初のラインで停止する（点Ｂ　１６６）。拡張ｓｔｅｐコマンド（「ｄｓｔｅｐ」コマンドと呼ぶ）は以下の２つの状況以外では、標準のｓｔｅｐコマンドと同じ作動を行う。

【００４８】「ｄｓｔｅｐ」コマンドがサーバでの呼出しをもたらす機能の呼出し時に実行された場合、次のソース・ラインがサーバで呼び出された機能の最初のラインになる。

【００４９】「ｄｓｔｅｐ」コマンドがサーバで呼び出された機能からの復帰時に実行された場合、次のソース・ラインは遠隔呼出しをもたらしたファンクション呼出し後にクライアントのソース・ラインとなる。ＤＯＥクライアントから見ると、「ｄｓｔｅｐ」コマンドはユーザがＩＤＬ生成コードから戻ってきたときに、このコマンドの特別な機能を開始する。ＤＯＥサーバから見ると、「ｄｓｔｅｐ」コマンドはユーザがＩＤＬ生成コードから戻ってきたときに、このコマンドの特別な機能を開始する。

【００５０】ｄｏｅＤｅｂｕｇｇｅｒを透過的に遮断することも必要である。遠隔デバッギングをサポートするのに何を行ったのかをユーザが知らないのであるから、ユーザがこれを取り消すことを期待すべきではない。

【００５１】「ｄｓｔｅｐ」コマンドが実行された後、ｄｏｅＤｅｂｕｇｇｅｒは現行機能が「ｒｅｍｏｔｅ　ｓｕｒｒｏｇａｔｅ　ｃｏｄｅ」であるかどうかを判定しようとする。「ｒｅｍｏｔｅ　ｓｕｒｒｏｇａｔｅ　ｃｏｄｅ」はＩＤＬ操作の遠隔呼出しを担当するコードである（すなわち、図８の項目１４８、１５０）。ＤＯＥシステムでは現在、「ｒｅｍｏｔｅ　ｓｕｒｒｏｇａｔｅ　ｃｏｄｅ」はすべてＩＤＬコンパイラによって生成されている。

【００５２】クライアント側では、ｄｏｅＤｅｂｕｇｇｅｒがＩＤＬ生成コードを下がっていっていることを認識する必要がある。これを達成するため、ＤＯＥはＩＤＬ生成コードの最初の層で使用される変数に特別な名前をつける。この変数が存在することが「ｄｓｔｅｐ」機能を開始するトリガとして役立つ。

【００５３】考えられるとおり、呼び出されるクライアント側の機能の名前は、その機能を実施するサーバ側の機能の名前と同じである。図８の例において、このことはプログラマがクライアント側で機能「ｆｏｏ」をコールし、自分がサーバ側で書いた機能「ｆｏｏ」を得ることを期待することに過ぎない。サーバは多くのクライアントにサービスを行えることがあり、したがって、その機能「ｆｏｏ」の多くの呼出しがサーバで生じることもある。サーバでの関連する呼出しは、サーバのどのスレッドがデバッグ対象クライアントのファンクション・コールに対応しているかを判定することによって特定される。２つの機能（クライアント側のものと、サーバ側のもの）をＤＯＥのメッセージ通過層に追加して、サーバの特定のスレッドの識別を援助する。

【００５４】あるオブジェクトに対するサーバを探し出すために、「ｆｉｎｄ　ｓｅｒｖｅｒ」機能がＤＯＥベース・クラスに追加されている。これは「ＧｅｔＩｍｐｌｅｍｅｎｔａｔｉｏｎ」という機能を実行する。この機能は任意のＤＯＥオブジェクトによって呼び出すことができ、サーバのホストｉｄおよびサーバのプロセスｉｄ（ｐｉｄ）を返すこととなる。ｄｏｅＤｅｂｕｇｇｅｒはクライアントの「ｆｉｎｄ　ｓｅｒｖｅｒ」機能を呼び出す。この機能はクライアントが使用しているオブジェクトに一部でなければならないから、ｄｏｅＤｅｂｕｇｇｅｒに組み込まれた機能ではない。「ｆｉｎｄ　ｓｅｒｖｅｒ」機能が呼び出された時点で、サーバが作動していなければ、サーバは起動される。

【００５５】上述のように、特別な変数が存在していることが、機能に取りかかるときに「ｄｓｔｅｐ」機能を呼び出さなければならないことを通知する「ｔｒｉｇｇｅｒ」として使用される。「ｄｓｔｅｐ」がサーバから戻る時期を通知するのには、次の２つの理由で、同様な「ｔｒｉｇｇｅｒ」は使用されない。

【００５６】トリガ機構が復帰時に利用できなかった。および

【００５７】呼び出された復帰がすでにわかっているコードを通って戻り（すなわち、このコードによってサーバに入った）、この知識を使って、復帰をトリガできた。

【００５８】サーバに初めて入ったとき、機能「ｆｏｏ」（図８のＩＤＬが生成した機能）についてのユーザのインプリメンテーションを直接呼び出す機能のスタック・ポインタ（復帰トリガと呼ぶ）がセーブされる。サーバからの復帰をチェックするときに、現行機能のスタック・ポインタを復帰トリガに関してチェックする。一致すれば、サーバから抜けることを継続する。

【００５９】ｄｏｅＤｅｂｕｇｇｅｒが初めてサーバに入るときには、新しいｄｂｘエンジンをサーバで起動しなければならない。このプロセスは「ＩｄｅｎｔｉｆｙＲｅｍｏｔｅＦｕｎｃｔｉｏｎ」プロセスの一部であり、遠隔ｄｂｘエンジンを識別し、ｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙオブジェクトを使用して遠隔ｄｂｘエンジンを起動／作成する拡張機能として追加されたものである。これを行う方法を図１４を参照して説明する。図１４はデバッガＧＵＩ５０２、ｄｂｘエンジン５０４、ヘルパ・プロセス５０６、クライアント側ラッパ・サーバ５１０およびクライアント５０８を有するローカル・ホスト５２０を示している。また、ｄｂｘエンジン５１２、ヘルパ・プロセス５１４、サーバ側ラッパ・サーバ５１８およびサーバ（呼び出された機能のインプリメンテーション）５１６を含んでいる遠隔ホスト５２２も示されている。デバッガＧＵＩへの接続およびサーバへの接続を含む新しいｄｂｘエンジンの初期化は、新しいｄｂｘエンジンを作成するコールを行うクライアント側ｄｂｘエンジン５０４によって達成される。

【００６０】ローカル・ホスト５２０のｄｂｘエンジン５０４はローカル・ホスト５２０のヘルパ・プロセス５０６を介したローカル・ホスト５２０のラッパ・サーバ５１０への要求によって、遠隔ホスト５２２にｄｂｘエンジン５１２を作成する。ヘルパ・プロセス５０６は、ラッパ・サーバ５１０と通信を行うＤＯＥアプリケーシ

ョンである。ｄｂｘエンジン自体がマルチスレッド安全（ＭＴセーフ）ではなく、ＤＯＥアプリケーションに作成することができない（すべてのＤＯＥアプリケーションは元来マルチスレッド化されている）ため、これが必要となる。ｄｂｘエンジンはヘルパ・プロセスを介してラッパ・サーバのサービスにアクセスする。ローカル・ホスト５２０のラッパ・サーバ５１０は遠隔ホスト５２２のラッパ・サーバにメッセージを送ってｄｂｘエンジンを遠隔サーバ５２２に作成し、サーバ５１６に接続するよう命令することを要求する。遠隔ホスト５２２にｄｂｘエンジンを作成するようにというローカル・ホスト５２０のｄｂｘエンジン５０４による要求は、遠隔ホスト５２２のｄｂｘエンジン５１２が完全に起動されるまで完了しない。遠隔ホスト５２２のラッパ・サーバは分岐し、遠隔ホスト５２２の新しいｄｂｘエンジン５１２を実行し、ｄｂｘエンジン５１２が終了するか、あるいはｄｂｘエンジン５１２が完全に起動されたことを示すメッセージを（ラッパ・サーバ５１８に）送るかするのを待つ。遠隔ホスト５２２のラッパ・サーバ５１８は２つのスレッドを作成する。一方のスレッドは分岐した子（ｄｂｘエンジン５１２）が終了するのを待機する。他方のスレッドは完全に起動されたｄｂｘエンジン５１２からのメッセージを待つ。これらのスレッドの一方がレジュームされると、子のスレッドは他方のスレッドを破壊し、適切な応答をローカル・ホスト５２０のラッパ・サーバ５１０に戻し、このラッパ・サーバは遠隔ホスト５２２でのｄｂｘエンジン５１２の作成要求を完了し、適切な状況値（作成成功または失敗）をローカル・ホストのｄｂｘエンジン５０４に返す。

【００６１】サーバの新しいｄｂｘエンジン５１２が起動した（上述のようにして）後、クライアントのｄｂｘエンジン５０４は新しいｄｂｘエンジン５１２にメッセージを送って、サーバ５１６に適切なブレークポイントをセットする。クライアントのｄｂｘエンジン５０４はサーバのｄｂｘエンジン５１２が完全に起動するまで（すなわち、デバッガＧＵＩとの接続を含む初期化を終わり、サーバに接続するまで）、このｄｂｘエンジン５１２にブレークポイント・メッセージを送ることができない。

【００６２】競合状態（レイス・コンディション）の可能性を回避するのに必要なｄｏｅＤｅｂｕｇｇｅｒの作動に関する付加的な事項を、ある程度詳細に説明する。

【００６３】「ｄｓｔｅｐ」中に行われるステップの記述はコマンドをサーバに接続されたｄｂｘエンジンに送って、サーバにブレークポイントをセットすることをいう。コマンドはサーバの適正なスレッドにブレークポイントをセットするのに十分な情報を含んでいる。ステップがトリガ機能に入ると、トリガ機能のコールで生じたコールを処理するサーバのスレッドを一意に識別できる情報はまだ存在していない。実際の機構はトリガ機能のコールによって生じる要求の識別子（要求ｉｄ）が利用できるポイントで、イベントがトランスポート層にセットされる。このイベントはコマンドをサーバのｄｂｘエンジンに送る。メッセージは要求ｉｄ、クライアントのホスト名、およびクライアントのプロセス間アドレスをを含んでおり、これが要求を一意に識別する。遠隔ホストのｄｂｘエンジンに送られたメッセージはサーバのメッセージ通過層にセットされるブレークポイント・セットをもたらす。このブレークポイントは要求ｉｄ、クライアントのホスト名、およびクライアントプロセス間アドレスの一致についてチェックし、一致が見つかった場合には、一致したスレッドがクライアントのトリガ機能によって開始された要求を処理するスレッドとなる。次いで、ブレークポイントが関数への進入時にそのスレッドにセットされ、サーバは継続される。

【００６４】クライアントのトリガ機能が「ｆｏｏ」である場合、ブレークポイントがセットされるサーバの機能も「ｆｏｏ」となる。しかしながら、クラス情報がサーバで利用できないので、「ｓｔｏｐ　ｉｎ　ｍｅｍｂｅｒ　ｆｏｏ」コマンドが特定のスレッドのチェックで使用される。場合によっては、自動的に生成されるＩＤＬコードに機能「ｆｏｏ」があり、生成コードを実行すると、ブレークポイントが決まる。ブレークポイントが決まると、ＩＤＬ生成コード内であるかどうかを判定するためのチェックが行われる。生成コード内にいる場合、他のブレークポイントがセットされ、実行が継続される。

【００６５】クライアントのトリガ機能が「ｆｏｏ」である場合、サーバのブレークポイントが「ｆｏｏ」になったとき、復帰トリガ・スタック・ポインタがセーブされる。復帰トリガは「ｄｓｔｅｐ」が「ｆｏｏ」の終わりで実行されたときに実行を継続するために使用される。ユーザは「ｃｏｎｔ」コマンドを使用して、サーバから明示的に継続することもできる。いずれの場合でも、セーブされた復帰トリガ・スタック・ポインタを廃棄し、これがサーバに対する他のコールで使用されないようにする。復帰時に実行されるように生成されたポイントで、イベントがトランスポート層にセットされ、セーブされた復帰トリガはそのイベントで廃棄される。イベントがスレッドでフィルタされるので、適正な要求の復帰だけが復帰トリガを廃棄する。

【００６６】図１４を参照して上述したように、メッセージをローカル・ホスト５２０のｄｂｘエンジン５０４から、遠隔ホスト５２２のｄｂｘエンジン５１２へ送って、サーバ５１６にブレークポイントをセットする。ブレークポイント・メッセージが送られた後、クライアント５０８が継続されるので、遠隔呼出しがクライアント５０８からサーバ５１６へ進む。しかしながら、ブレークポイント・メッセージを遠隔ホスト５２２のｄｂｘエンジン５１２が受け取って、処理してから、遠隔呼出し

がサーバ５１６に到達するという保証はない。ブレークポイントが実際にセットされてから、遠隔呼出しが行われることを保証するために、ローカル・ホスト５２０のｄｂｘエンジン５０４は停止し、遠隔ホスト５２２のｄｂｘエンジン５１２からのメッセージを待つ。遠隔ホスト５２２のｄｂｘエンジン５１２はサーバ５１６にブレークポイントをセットすると、メッセージをローカル・ホスト５２０のｄｂｘエンジン５０４に送って、クライアント５０８を継続する。

【００６７】上記で明らかにした付加的な変更／拡張を説明する。

【００６８】あるｄｂｘエンジンが他のｄｂｘエンジンと通信することを可能とする変更は、メッセージを他のｄｂｘエンジン（ターゲットｄｂｘエンジンと呼ぶ）へ送るために、特定のＴｏｏｌＴａｌｋメッセージ（「ｒｃｍｄ」メッセージと呼ぶ）をｄｂｘエンジンからデバッガＧＵＩに渡すことができるようにするデバッガＧＵＩおよびｄｂｘエンジンの両方に対する変更を含んでいる。デバッガＧＵＩはユーザからのコマンドを受け入れ、これを特定のｄｂｘエンジンへ送る。「ｒｃｍｄ」メッセージはターゲットｄｂｘエンジンが作動しているホストマシンの名前、ターゲットｄｂｘエンジンがデバッグしているプロセスのプロセス識別子（ｐｉｄ）、およびターゲットｄｂｘエンジンに対するメッセージを含んでいる。デバッガＧＵＩはｄｂｘエンジンのリストを維持しており、このリストはｄｂｘエンジンが作動しているホストの名前、およびデバッグされているプロセスのｐｉｄを含んでいる。デバッガＧＵＩは「ｒｃｍｄ」メッセージを取得すると、指定されたホストマシンで作動しており、所与のｐｉｄを有するプロセスをデバッグしているｄｂｘエンジンに対するｄｂｘエンジンのリストに到達する。ターゲットｄｂｘエンジンに対するメッセージは次いで、そのｄｂｘエンジンに送り出される。

【００６９】デバッガＧＵＩが特定のｄｂｘエンジンにフォーカスするのを可能とする変更は、デバッガＧＵＩに送信ｄｂｘエンジンに対するアテンションのフォーカスを変更させるために、特定のＴｏｏｌＴａｌｋメッセージ（「ａｔｔｅｎｔｉｏｎ」メッセージと呼ぶ）をｄｂｘエンジンからデバッガＧＵＩに渡すことができるようにするデバッガＧＵＩおよびｄｂｘエンジンの両方に対する変更を含んでいる。デバッガＧＵＩは特定のｄｂｘエンジンに関連する表示を維持している。たとえば、特定のｄｂｘエンジンがデバッグしているプロセスに対するソース・プログラムがデバッガＧＵＩによって表示される。デバッガＧＵＩは１組の表示を有しており、一度に１つのｄｂｘエンジンからの情報を示すことができる。「ａｔｔｅｎｔｉｏｎ」メッセージはデバッガＧＵＩがその表示を現行ｄｂｘエンジンに対する情報から「ａｔｔｅｎｔｉｏｎ」メッセージを送っているｄｂｘエンジンの情報に変更することを示す。

【００７０】デバッガＧＵＩからすべての活動ｄｂｘエンジンのリストを取得できるようにするのに必要な変更は、デバッガＧＵＩに各ｄｂｘエンジンが作動しているホスト名、およびそのｄｂｘエンジンがデバッグしているプロセスのプロセス識別子（ｐｉｄ）のリストを返させるために、特定のＴｏｏｌＴａｌｋメッセージ（「ｇｅｔ　ｄｂｘ　ｅｎｇｉｎｅｓ」メッセージと呼ぶ）をｄｂｘエンジンからデバッガＧＵＩに渡すことができるようにするデバッガＧＵＩおよびｄｂｘエンジンの両方に対する変更を含んでいる。デバッガＧＵＩはｄｂｘエンジンが作動しているホストの名前、およびそのｄｂｘエンジンがデバッグしているプロセスのｐｉｄを含んでいるすべてのｄｂｘエンジンのリストを維持している。デバッガＧＵＩは「ｇｅｔ　ｄｂｘ　ｅｎｇｉｎｅｓ」メッセージを受け取ると、ホストの名前およびデバッグされているプロセスのｐｉｄを各ｄｂｘエンジンに関して実行し、それをｇｅｔ　ｄｂｘ　ｅｎｇｉｎｅｓメッセージを送ったｄｂｘエンジンに返送する。

【００７１】ＩＶ．分散デバッガー使用方法

本発明のｄｏｅＤｅｂｕｇｇｅｒを作成するのに必要なＳＰＡＲＣｗｏｒｋｓデバッガ・システムに対する変更／拡張を説明してきたが、ｄｏｅＤｅｂｕｇｇｅｒを使用する方法をここで説明する。

【００７２】図１０を参照して、ｄｏｅＤｅｂｕｇｇｅｒの作動２２０を説明する。まず、ローカルマシンのプログラマ／開発者はｄｏｅＤｅｂｕｇｇｅｒを起動する（２２２）。ターゲット・プログラムが示され（２２４）、「ｄｓｔｅｐ」コマンドが希望する機能に対して指定される（２２６）。ｄｏｅＤｅｂｕｇｇｅｒは標準「ｓｔｅｐ」コマンドを実行し（２２８）、ｄｏｅＤｅｂｕｇｇｅｒはターゲット・インプリメンテーションがローカルか遠隔かを決定しようと試みる（２３０）。ターゲットが遠隔であると認識すると（ＩＤＬが生成した「ｒｅｍｏｔｅｓｕｒｒｏｇａｔｅ　ｃｏｄｅ」を認識することによって）、「ｆｉｎｄ　ｓｅｒｖｅｒ」機能を実行して（２３２）、ターゲット・インプリメンテーションのホストｉｄおよびｐｉｄを見つけだす。ローカルｄｂｘエンジンはコマンドを発行して、見つかったホストにｄｂｘエンジンを作成し（２３４）、ブロックし、応答を待つ。見つかったホストはターゲット・インプリメンテーションに接続されているｄｂｘエンジンがあるかどうかを判定する（２３６）。すでにｄｂｘエンジンが作動している場合には（２５０）、見つかったホストのサーバは復帰メッセージを呼出し元クライアント側ｄｂｘエンジンに送る（２５２）。クライアント側ｄｂｘエンジンはメッセージを受け取って、ブロックを解除する（２５４）。クライアント側エンジンは次いで、メッセージをサーバのｄｂｘエンジンに送って、指定された機能に一時ブレークポイントをセットする（２６２）（図１１において）。図１１を続けると、サーバｄ

ｂｘエンジンはコマンドを実行して、ターゲット機能にブレークポイントをセットし（２６４）、復帰トリガ・スタック・ポインタをセーブする（２６６）。サーバｄｂｘエンジンはターゲット・インプリメンテーションを「継続」する（２６８）。その後、ターゲット・インプリメンテーションは指定されたブレークポイントにヒットし（２７０）、サーバｄｂｘエンジンはブレークポイントを処理し、クライアント・ホストのデバッガＧＵＩにメッセージを送って、サーバｄｂｘエンジンにフォーカスする（２７２）。デバッガＧＵＩを使用しているプログラマはこれで、遠隔機能をこれがあたかもクライアント・ホストにあるような状態でデバッグできるようになる（２７４）。その後、システムはデバッグ・セッションが終わったかどうか（すなわち、ｑｕｉｔコマンドを受け取ったかどうか）を調べるためにチェックを行い（２７６）、終わっていない場合には（２７８）、デバッグ・セッションを継続する（２８２）。ｑｕｉｔコマンドを受け取っている場合には（２８０）、遠隔ｄｂｘエンジンは終了し、ターゲット・プロセスを接続解除し（２８４）、セッションを終わらせる（２８６）。

【００７３】図１０のブロック２３６に戻って、見つかったサーバ・ホストで作動しているｄｂｘエンジンがない場合には（２３８）、サーバはクライアント・ホストのｄｂｘエンジンに、ｄｂｘエンジンが作動していないことを伝える（２４０）。ヘルパ・オブジェクトを使用しているクライアントｄｂｘエンジンはクライアント側ｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙに対する要求を処理し、見つかったホストのサーバにｄｂｘエンジンを作成する（２４２）。クライアント側ｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙオブジェクトはサーバ側ｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙインプリメンテーションをコールし（２４４）、ｄｂｘエンジンが見つかったホストのサーバに作成され、ターゲット機能のインプリメンテーションに接続する（２４６）。作成され、ターゲット機能に接続したばかりのこのｄｂｘエンジンが起動され（２４８）、復帰メッセージがクライアント側ｄｂｘエンジンに送られ（２５２）、ブロック２５４および図１１のブロックに関して上述したように、プロセスがこのポイントから継続する。

【００７４】図１０のブロック２４６に示した「遠隔ｄｂｘエンジン作成および接続」プロセスを図１２を参照して詳細に説明する。図１２において、「作成」プロセスが開始され（３０２）、クライアント側ｄｂｘエンジンがローカル・ヘルパ・プロセスをコールする（３０４）。ローカル・ヘルパ・プロセスは「ｃｒｅａｔｅ」コマンドをｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙオブジェクトに対して発行する（３０６）。ｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙオブジェクトはメッセージを見つかったホスト３０８のｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙインプリメンテーションに送り、このインプリメンテーションは「ｃｒｅａｔｅ」コマンドを実行する（３１０）。ｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙインプリメンテーションはｄｂｘエンジンに対して「ｆｏｒｋ」および「ｅｘｅｃ」を行い（３１６）、新たに作成されたｄｂｘエンジンからのメッセージを待つ（３１８）。新しいｄｂｘエンジンが何らかの理由で完全に起動されていない場合（３２４）、「Ｆａｉｌｅｄ＿ｔｏ＿ｓｔａｒｔ」メッセージが返され（３２６）、ｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙインプリメンテーションは終了する（３２０）。新しいｄｂｘエンジンが完全に起動されている場合には（３２２）、新しいｄｂｘエンジンはターゲット機能に接続するよう指示される（３２８）。新しいｄｂｘエンジンが接続できない場合には（３３４）、メッセージ「Ｆａｉｌｅｄ＿ｔｏ＿ａｔｔａｃｈ」が返され（３３６）、ｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙインプリメンテーションは終了する（３２０）。新しいｄｂｘエンジンが接続できる場合には（３３２）、メッセージ「Ａｔｔａｃｈｅｄ＿ａｎｄ＿ｒｕｎｎｉｎｇ」が返され、ｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙインプリメンテーションは終了する（３２０）。ターゲット・サーバへの「接続の失敗」が要件に接続するそれ自体の許可を有しているサーバによって生じることがあり、この場合、接続のための他の機構が必要となることがあることに留意されたい。

【００７５】図１１のブロック２８４に示した「遠隔ｄｂｘエンジン終了および接続解除」プロセスを、図１３を参照して詳細に説明する。図１３において、プログラマ／開発者がデバッガＧＵＩに対して「Ｑｕｉｔ」コマンドを発行し（４０２）、デバッガＧＵＩがこのコマンドをローカルｄｂｘエンジンに渡すことによって「終了」プロセスが開始される（４０４）。ローカルｄｂｘエンジンは「ＱｕｉｔＳｅｓｓｉｏｎ」メッセージをヘルパ・プロセスを介してｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙに送り、デバッグ・セッションを終了する（４０６）。「ＱｕｉｔＳｅｓｓｉｏｎ」コマンドにより、関与しているホストの各々のｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙオブジェクトは各ｄｂｘエンジンに信号を送るが（４０８）、このｄｂｘエンジンはデバッグ・セッションの一部である（４１０）。各ｄｂｘエンジンは送られた信号に対する信号ハンドラを有しており、このハンドラは信号がｄｂｘＷｒａｐｐｅｒＦａｃｔｏｒｙから送られているのかどうかを調べ、そうである場合には、ｄｂｘエンジンはデバッグしているプロセスとの接続を解除し（４１２）、終了する。他の実施の形態はｄｂｘエンジンをそのターゲット機能から接続解除するステップ（４１４）や、「ｄｂｘ＿ｄｅｂｕｇｇｅｒ＿ｄｅｔａｃｈｅｄ＿ａｎｄ＿ｄｅｌｅｔｅｄ」などのメッセージをクライアント側ｄｂｘエンジンに返すステップ（４１６）などの付加的なステップを含んでいることができる。

【００７６】ｄｂｘＷｒａｐｐｅｒを実施するインタフェースを図１５に示す。当分野の技術には、各種のインプリメンテーションがこの機能に対して行えることが認識されよう。

【００７７】本発明を特定のオペレーティング・システム、プログラム・コード、ならびにオブジェクトおよびオブジェクト・リファレンスの定義に関して説明してきたが、本発明を所与のオペレーティング環境内の、あるいは異なるオペレーティング・システムまたはオブジェクト・システム環境の各種の変形のうちの任意のもので実施できることが当分野の技術者には理解されよう。同様に、図示の特定のクライアントおよびサーバの構成または組合せは、本発明を使用することのできるクライアントとサーバの多くのこのような構成、ならびにオブジェクトとサブオブジェクトの関係の代表的なものに過ぎない。さらに、図面が説明のためのものに過ぎず、本発明の限定事項と取るべきではないことが理解されよう。遠隔ｄｂｘエンジンの他の機能を備えたクライアント側デバッガＧＵＩとの付加的な組合せは、ターゲット・オブジェクトにフレンドリ・ユーザ・インタフェースをもたらすグラフィカル・ユーザ・インタフェース（ＧＵＩ）エージェントとのｄｂｘエンジンの組合せ、ユーザの既知の選択に基づいて遠隔要求を変更する人工知能エージェントとの遠隔ｄｂｘエンジンの組合せ、遠隔要求に対する回答をキャッシュするキャッシュ・プログラムとの遠隔ｄｂｘエンジンの組合せ、数人のユーザからの入力をマージして、これをターゲットに送るテレコンファレンス・アプリケーションとの遠隔ｄｂｘエンジンの組合せ、あるいはマルチメディア・システムの多数のオーディオおよびビデオ・アクセス・エージェントとの遠隔ｄｂｘエンジンの組合せを含んでいる。これらの考えられるｄｂｘエンジンとデバッガＧＵＩの組合せは本明細書で開示した遠隔デバッガ機能の可能な用途を限定することを意図したものではなく、当分野の技術者が例と認識する例を表しているに過ぎない。ｄｏｅＤｅｂｕｇｇｅｒの発明の範囲は、したがって、首記の特許請求の範囲、ならびに特許請求の範囲の対象となる同等物の全範囲を参照することによって決定されるものである。

【図面の簡単な説明】

【図１】汎用コンピュータおよび関連装置を示す図である。

【図２】分散コンピュータ・システムを示す図である。

【図３】ユーザ、クライアント・アプリケーション、オブジェクト・リファレンス、オブジェクト・インプリメンテーション、およびオブジェクト・リファレンス作成プログラムの関係を示す、複数のマシンを備えたクライアント・サーバ・システム構成の図である。

【図４】単一のマシンを使用したクライアント・サーバ構成の図である。

【図５】ＳＰＡＲＣｗｏｒｋｓデバッガを示す図である。

【図６】クライアント・アプリケーション、オブジェクト・リファレンスおよびＳＰＡＲＣｗｏｒｋｓデバッガの間の関係を示す図である。

【図７】複数のサーバにアクセスする必要のある分散オブジェクト環境（ＤＯＥ）クライアント・アプリケーションの例を示す図である。

【図８】ユーザ・コードとＤＯＥクライアントまたはサーバの基礎ＤＯＥ機構の間の分割を示す図である。

【図９】ｄｏｅＤｅｂｕｇｇｅｒ構成を示す図である。

【図１０】分散環境のｄｏｅＤｅｂｕｇｇｅｒの作動を示す流れ図である。

【図１１】ｄｂｘエンジン間プロセスを示す流れ図である。

【図１２】ｄｏｅＤｅｂｕｇｇｅｒ作成および接続プロセス（図１０の流れ図のブロック２４６）の作動を示す流れ図である。

【図１３】ｄｏｅＤｅｂｕｇｇｅｒ終了および接続解除プロセス（図１１の流れ図のブロック２８４）の作動を示す流れ図である。

【図１４】ｄｏｅＤｅｂｕｇｇｅｒ、ｄｂｘエンジン、ラッパ・サービスおよびクライアント・アプリケーションおよびサーバ・オブジェクト・インプリメンテーションの間の関係を示す図である。

【図１５】ｄｂｘＷｒａｐｐｅｒの典型的なインタフェースを示す図である。

【符号の説明】

１　コンピュータ、　２　入出力（「Ｉ／Ｏ」）部
３　中央処理装置（「ＣＰＵ」）、　４　メモリ部
５　キーボード、　６　表示装置
７　ＣＤ－ＲＯＭ駆動装置、　８　ＣＤ－ＲＯＭ媒体
９　ディスク記憶装置、　１０　プログラム
２０、２２、２４　コンピュータ
２６　通信リンク
２８　共用メモリ装置
３０　ユーザ
３４　クライアント・アプリケーション
３６　オブジェクト・リファレンス
４６　オブジェクト・インプリメンテーション
５０　サーバ
９２　ホストマシン
９６　ＴｏｏｌＴａｌｋ通信リンク
９８　ｄｂｘエンジン
１１２　赤ホスト
１３０　白ホスト
１４２　ローカル・ホスト
１４４　機能「ｆｏｏ」
１４８　生成コード

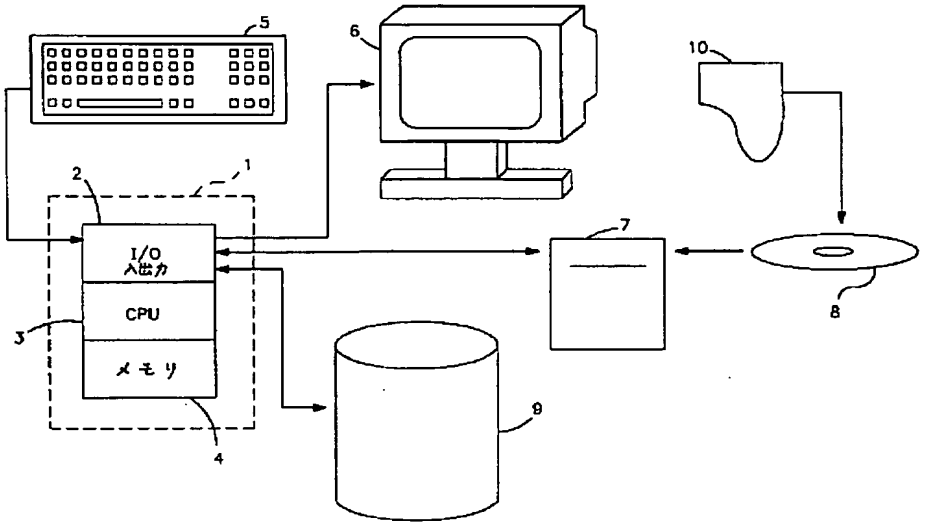
31

５０２　デバッガＧＵＩ
５０６　ヘルパ・プロセス

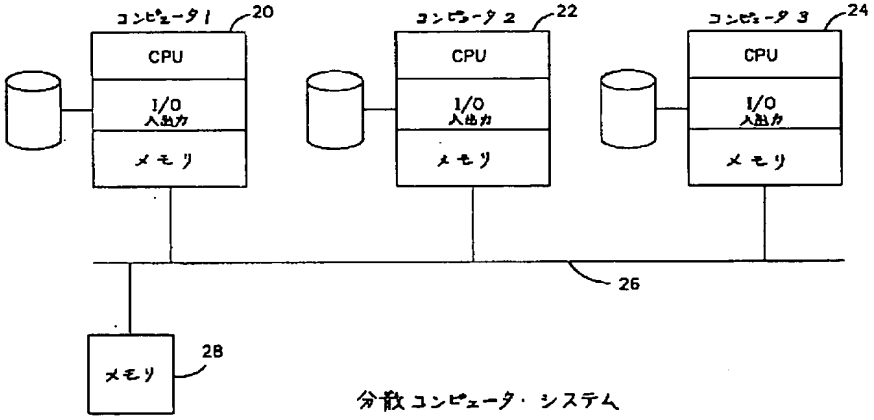
32

５１０　ラッパ・サーバ
５２２　遠隔ホスト

【図１】

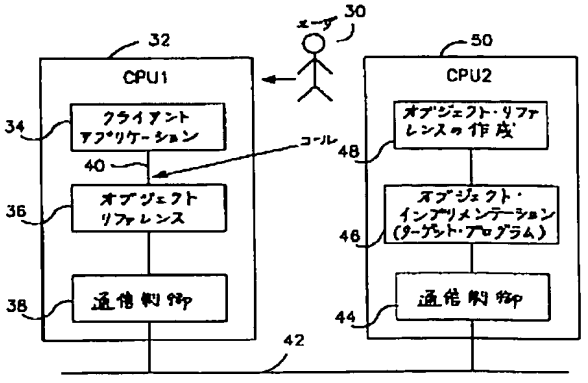


【図２】

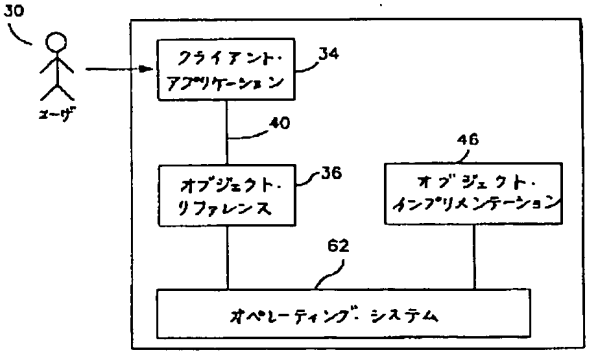


分散コンピュータ・システム

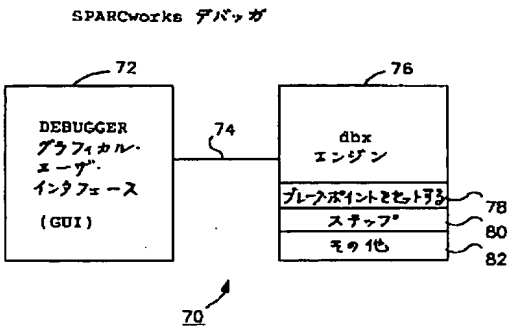
【図３】



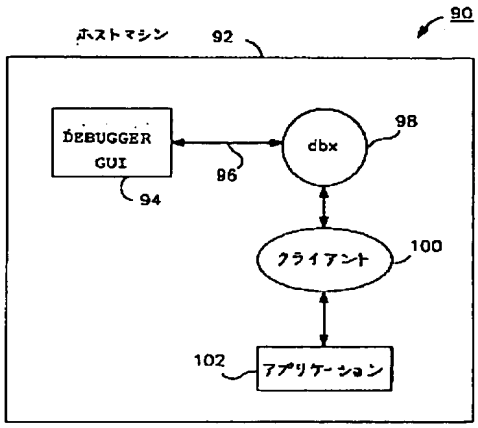
【図４】



【図５】

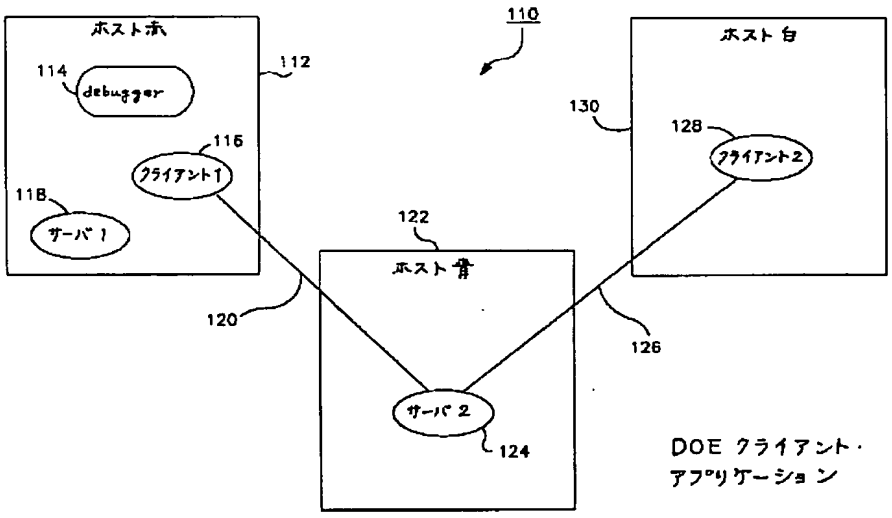


【図６】

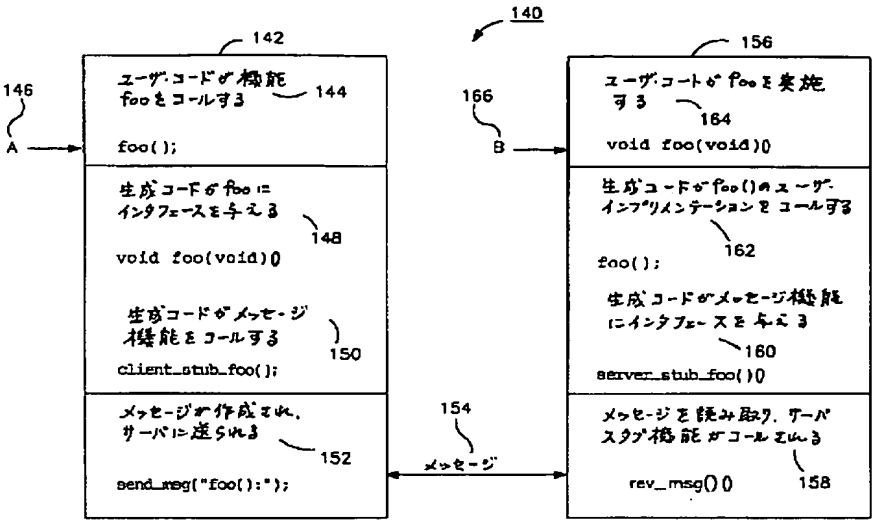


基本的な Debugger の作動

【図７】

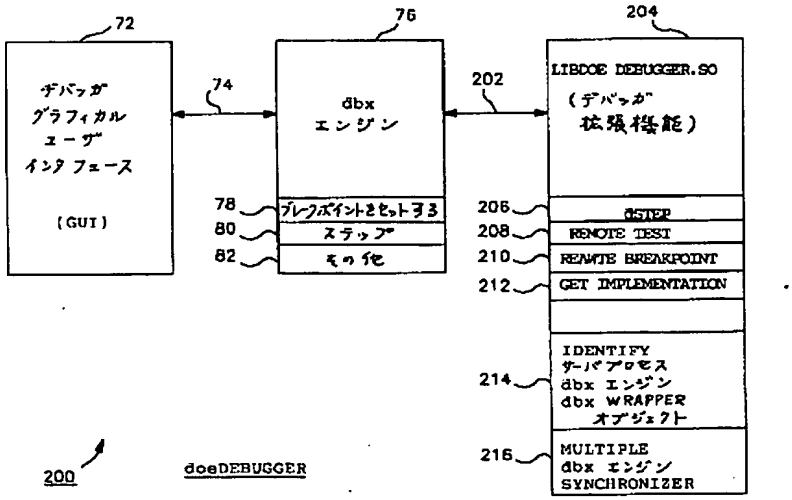


【図８】



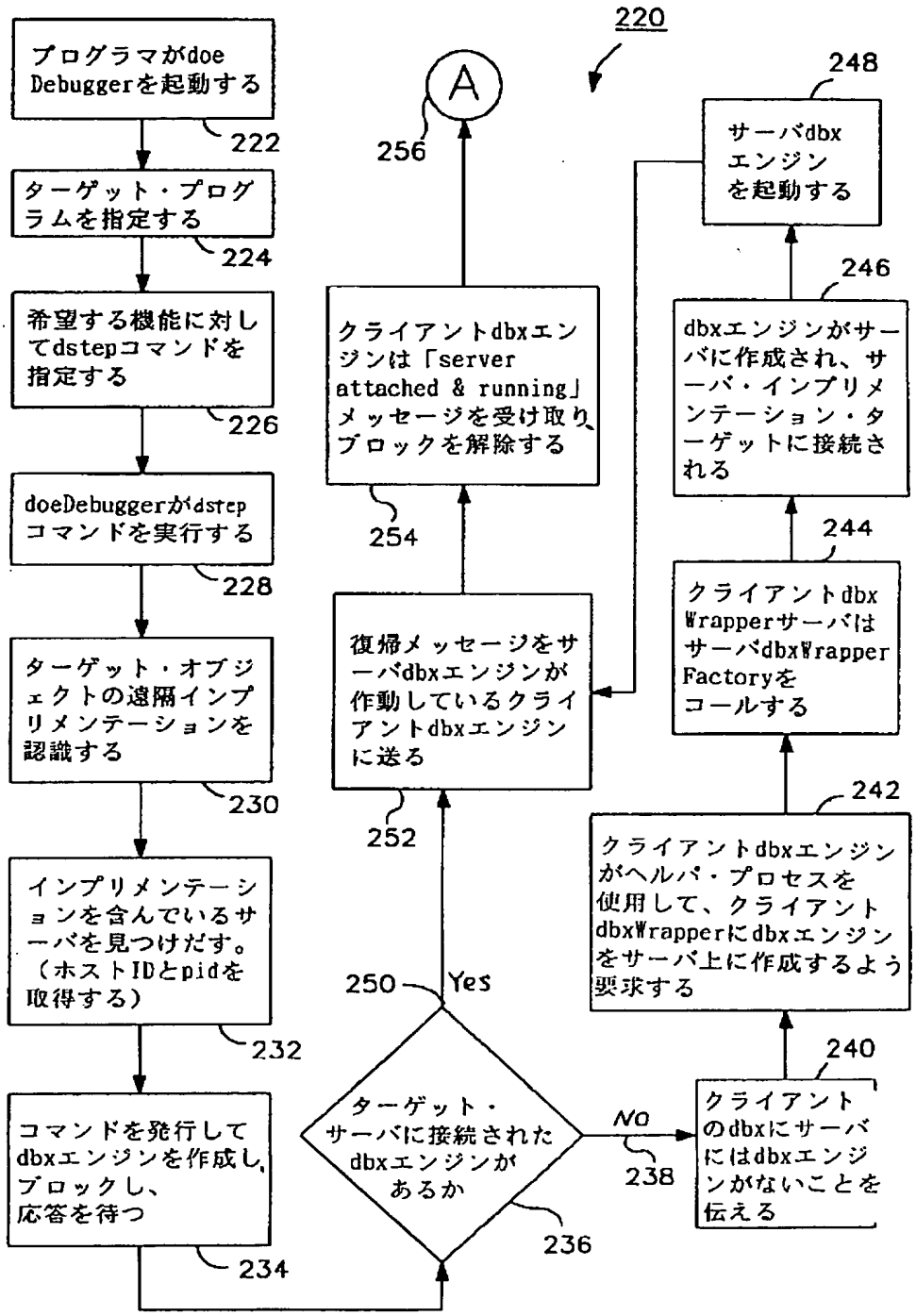
ユーザ と サポート・コード

【図９】

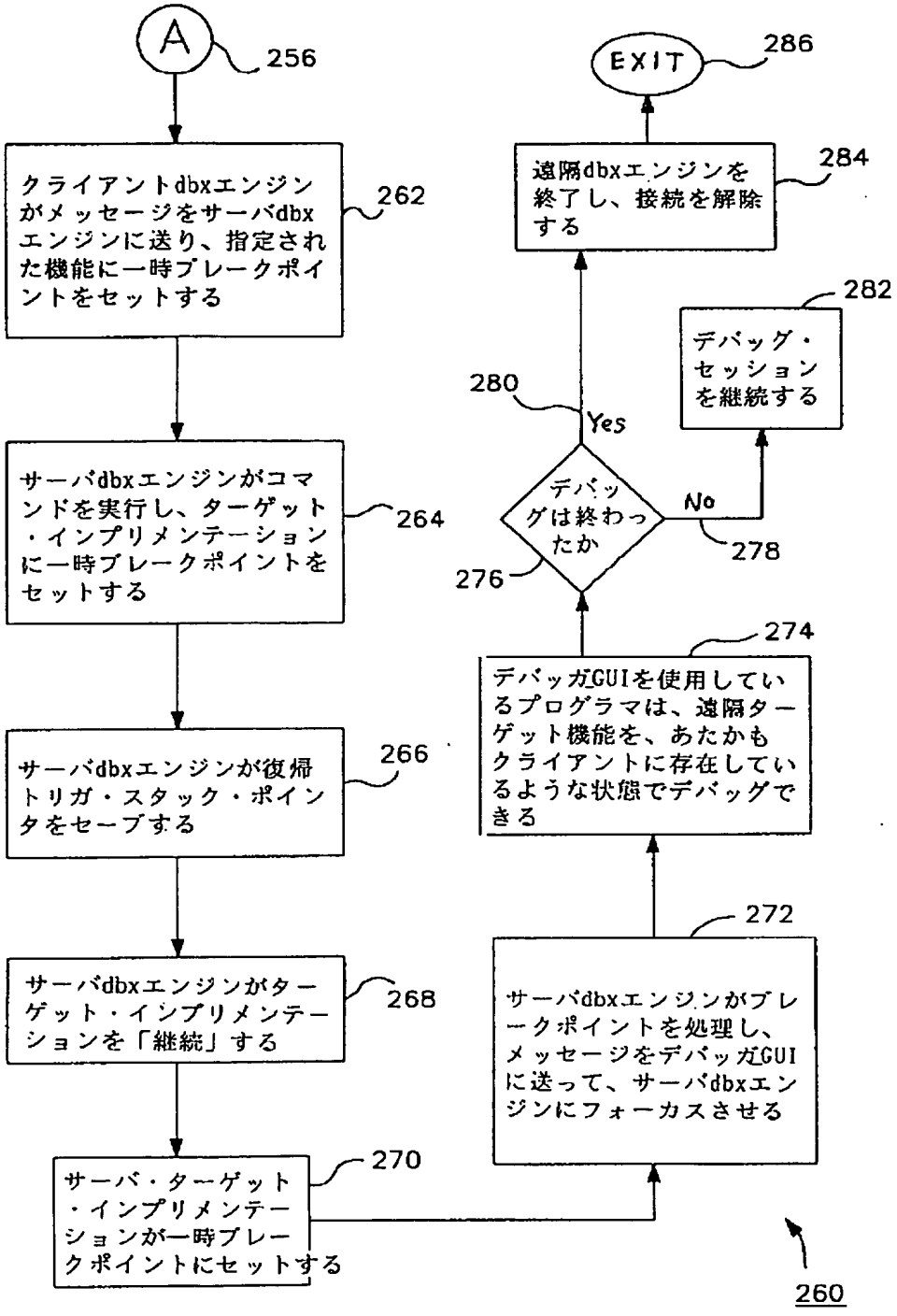


【図１０】

doe Debuggerの作動

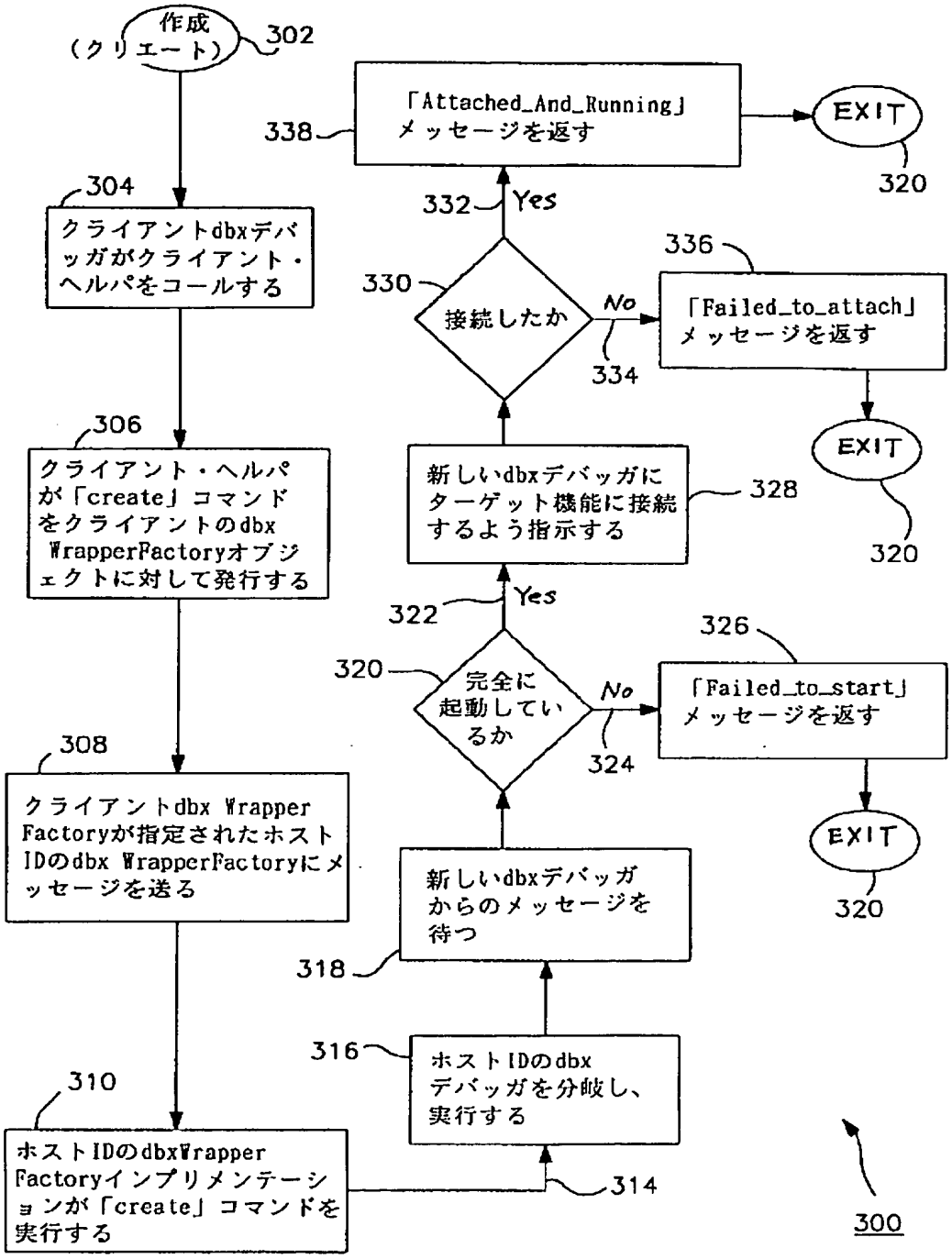


【図１１】


A
256
クライアントdbxエンジンがメッセージをサーバdbxエンジンに送り、指定された機能に一時ブレークポイントをセットする
262
サーバdbxエンジンがコマンドを実行し、ターゲット・インプリメンテーションに一時ブレークポイントをセットする
264
サーバdbxエンジンが復帰トリガ・スタック・ポインタをセーブする
266
サーバdbxエンジンがターゲット・インプリメンテーションを「継続」する
268
サーバ・ターゲット・インプリメンテーションが一時ブレークポイントにセットする
270
サーバdbxエンジンがブレークポイントを処理し、メッセージをデバッガGUIに送って、サーバdbxエンジンにフォーカスさせる
272
デバッガGUIを使用しているプログラマは、遠隔ターゲット機能を、あたかもクライアントに存在しているような状態でデバッグできる
274
デバッグは終わったか
276
Yes
280
No
278
デバッグ・セッションを継続する
282
遠隔dbxエンジンを終了し、接続を解除する
284
EXIT
286
260

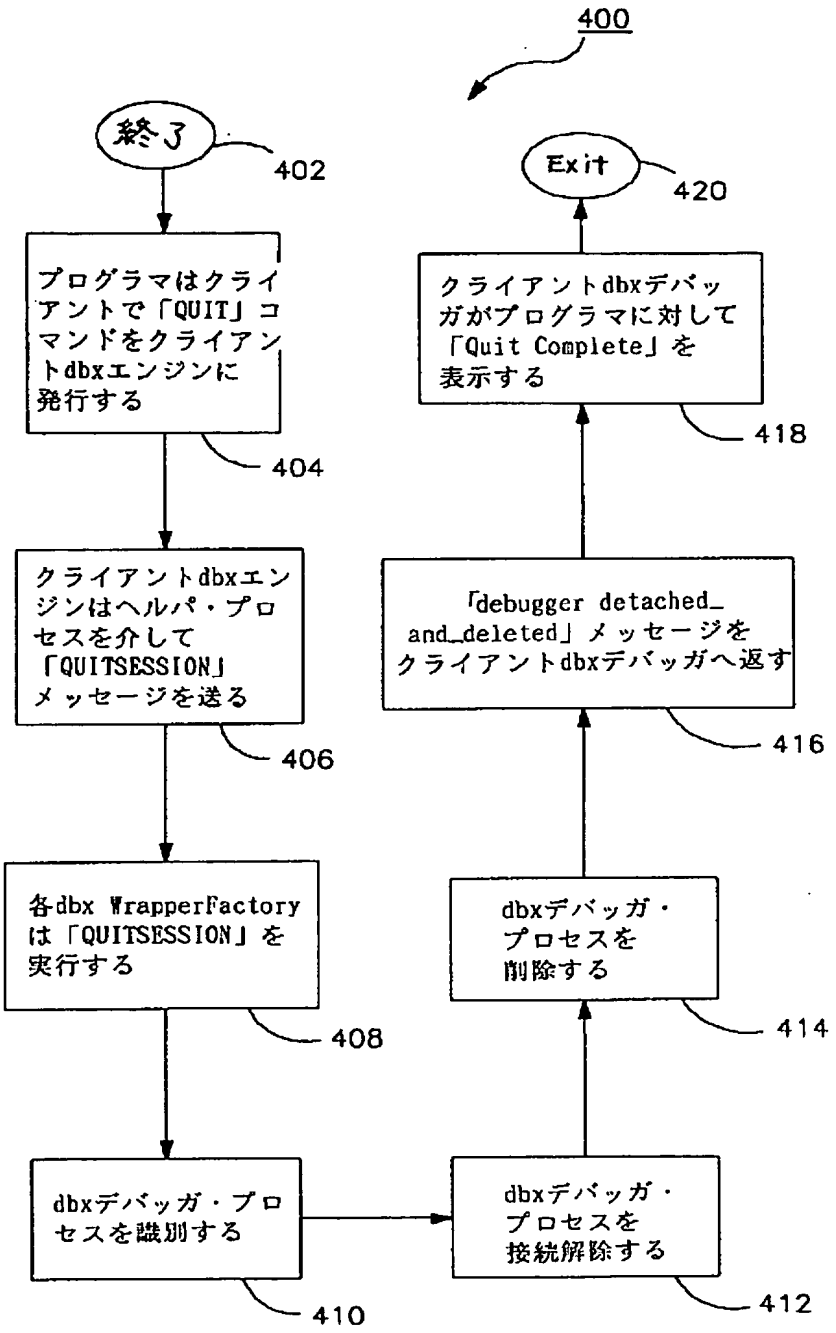
【図１２】

遠隔dbxエンジンの作成と接続

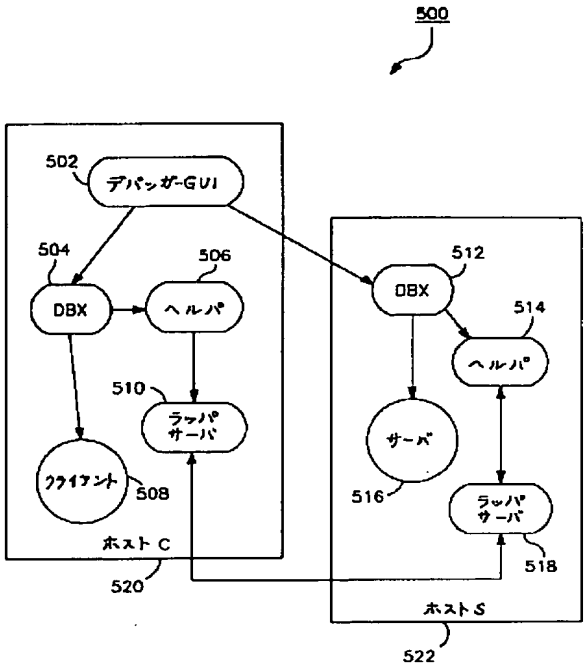


【図１３】

遠隔dbxエンジンの終了と接続解除



【図１４】



【図１５】

DbxWrapper インタフェース

```
enum DbxWrapperFailedReasons {
   FAILED_TO_START,
   FAILED_TO_ATTACH,
   ALREADY_BEING_DEBUGGED,
   PERMISSION_DENIED

};

exception DbxWrapperFailed {
   DbxWrapperFailedReason reason;

}

interface DbxWrapper {
   void SynchronizeDbxEngine();
   void Quit();

};

interface DbxWrapperFactory{

   DbxWrapper Create(
         in object servant,      // デバッグするオブジェクト
         in string path,         // dbx プログラム パス
         in string dbxEngine,    // dbx プログラム名
         in string display,      // メディスプレイが接続する
         in string arguments     // tool Talk アドレスを含む
      )raises(DbxWrapperfailed);

   void DbxEngineAttached();

};
```

フロントページの続き

(72)発明者　ジョン・エイ・マサミツ
アメリカ合衆国　94550　カリフォルニア
州・リバモア・クリーク　ロード・1873